ADVANCES IN HIERARCHICAL REAL-TIME SYSTEMS: INCREMENTALITY, OPTIMALITY, AND MULTIPROCESSOR CLUSTERING

Arvind Easwaran

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial

Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2008

Insup Lee
Oleg Sokolsky
Supervisors of Dissertation

Rajeev Alur
Graduate Group Chairperson

COPYRIGHT

Arvind Easwaran

2008

To Mom and Dad

Acknowledgments

Foremost, I would like to thank my advisors, Insup Lee and Oleg Sokolsky, for the unstinting support they have given me during my PhD studies. Insup introduced me to various fields of study, and gave me the academic freedom necessary to successfully complete my dissertation. His ability to focus on the problem as a whole without being distracted by details has helped me out of many a dead-ends, and has had a tremendous impact on the overall quality of this work. Oleg's keen eye for details and the various discussions we had over years, have also contributed immensely to the success of this dissertation.

I am also grateful to my dissertation committee for the valuable suggestions that I received throughout the process. Interesting questions raised during the proposal by the committee chair Sampath Kannan, resulted in our work towards optimal interfaces for hierarchical systems in Chapter 6. I have also learnt many finer skills of academic research from Sampath during my early years at Penn. I am eternally grateful to Sanjeev Khanna for giving me the opportunity to conduct research at Penn and also for all the help on my dissertation, including the computation of lower bound for incremental analysis in Chapter 4. A big thanks also to Sanjoy Baruah, my external committee member, for patiently answering all my queries and for some invaluable suggestions in the process. Without Sanjoy's seminal contributions to various results in scheduling theory, this dissertation would be grossly incomplete. Boon Thau Loo has been an inspiration ever since I was a teaching assistant for his course in Spring 2008. His enthusiasm and interest in

my work have played a significant role in finishing this dissertation.

Insik Shin and Madhukar Anand have been the non-faculty pillars of this dissertation. Insik introduced me to the world of hierarchical scheduling, and ever since has played a major role in my PhD studies. His knowledge in this research area and the ability to identify interesting problems have helped my research immensely. Madhukar's strong quantitative skills have also contributed towards many results that form the cornerstone of this dissertation. Beyond research, I am grateful to Mike Felker for giving me the illusion that bureaucracy is non-existent in Penn. A big thanks also to my various roommates and friends at Penn, for making my non-academic life fun enough to be able to withstand the idiosyncrasies of research. I cherish all the wonderful moments with friends like Arvind Bhusnurmath, Aneesh Kapur, Prashant Menon, Nahush Patel, and Arshad Karim, to name a few. The vibrant and diverse environment in Penn and in Philadelphia also occupy special places in my memory.

Finally, but also the most, I am indebted to my family for nurturing and whole-heartedly supporting my ambitions in life. Thank you mom and dad for steadfastly believing in my goals, and for enabling me to achieve them. A big hug to my sister for nudging me on during my studies at Penn, and also for livening up the numerous phone conversations and vacations. A special thanks to my girlfriend, Abhiruchi Gadgil, for making my last three years at Penn the most fun-filled and for infusing all the positive energy, pushing my research towards completion.

ABSTRACT

ADVANCES IN HIERARCHICAL REAL-TIME SYSTEMS: INCREMENTALITY, OPTIMALITY, AND MULTIPROCESSOR CLUSTERING

Arvind Easwaran

Supervisors: Insup Lee and Oleg Sokolsky

Component-based engineering is a popular design strategy for multi-functional and complex real-time systems; it decomposes the system into simpler components and then composes them using interfaces that hide internal complexities. Since components also have time-constrained resource demand, this design leads to hierarchical real-time systems that share resources under a scheduling hierarchy. Due to resource-constrained operating environments, schedulability analysis of such hierarchical systems – analysis to verify satisfaction of component timing constraints – is an important problem that must be addressed in the design phase. Additionally, to preserve the principles of component-based engineering, this analysis must be compositional, *i.e.*, system schedulability must be determined by composing interfaces that abstractly represent component resource requirements.

This dissertation addresses two problems pertaining to compositional schedulability analysis. First, it develops analysis techniques that are also incremental; analysis is independent of the order in which component interfaces are composed. This allows interfaces to be reused for efficient, on-the-fly verification of changes to components. Second, it addresses the problem of resource wastage in compositional analysis. It develops a new component interface whose resource utilization is better than existing interfaces. It also characterizes the notion of optimal resource utilization for hierarchical systems, and determines conditions under which this optimality can be achieved. Finally, it demonstrates the effectiveness of these techniques on data sets obtained from avionics systems.

With increasing popularity of multiprocessor-based embedded real-time systems, development of scheduling algorithms for multiprocessors is gaining importance. Algorithms that manage platform parallelism either by splitting it into multiple uniprocessors (partitioned scheduling), or by allowing demand to freely migrate across the platform (global scheduling), have been developed. In-spite of a reasonable success with these algorithms, many scheduling problems still remain open. Towards addressing them, this dissertation proposes a novel scheduling technique that virtualizes demand allocation through a two-level scheduling hierarchy. This technique supports more general demand distributions across the platform in comparison to existing algorithms, and hence provides a finer control over platform parallelism.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Component-based real-time systems

Nowadays software systems find application in various domains where time is of the essence. For example, when a pilot issues a command to land an aircraft, not only should the software system in the aircraft respond correctly, but it should do so in a timely manner. Such systems in which the software is expected to execute within pre-specified temporal deadlines are called *Real-Time* systems. In the case of air-crafts, failure to meet some timing requirements may even lead to loss of lives, and therefore it is essential to meet those deadlines. Such timing requirements are called *hard* temporal deadlines. There also exist deadlines in real-time systems whose compliance is not required for system safety, but satisfying them adds value to the system. For example, in highly interactive systems like web-services, a shorter response time is desirable to end users. Timing requirements in such systems are called *soft* temporal deadlines. Other applications of real-time systems include automobile, medical devices, factory automation, etc.

Real-time systems are frequently characterized using two attributes; *workload* and *scheduling algorithm*. The workload defines timing requirements in terms of what are the deadlines and how much resource is required within those deadlines. Many abstract models are used for specifying

this workload. Periodic [82], sporadic [88], multi-frame [89, 24], branching [18, 17], etc., are different types of such models also called *tasks*. For example, a periodic task is defined using three parameters: period of recurrence (T), worst-case execution time (C), and deadline (D). Each instance of this task requires C units of resource within D time units of its release. Furthermore, instances of this task are released periodically with a period of T time units. Sporadic tasks are similar to periodic tasks, except that period of recurrence now denotes the minimum separation, instead of exact separation, between release times of successive instances. Whenever the workload of a real-time system is specified as a collection of one or more such tasks, we refer to the system as an *elementary real-time system* in this dissertation.

Another attribute that characterizes an elementary system is the *scheduling algorithm*. In order to prioritize resource allocation, this algorithm defines an ordering between tasks in a system's workload. Dynamic-priority scheduling algorithms such as Earliest Deadline First (EDF) [82] and Least Laxity First [88], assign priorities to task instances that vary with time. For example under EDF, at any point in time, the task instance which has the earliest deadline also has the highest priority. On the other hand, fixed-priority scheduling algorithms such as Rate Monotonic (RM) [82] and Deadline Monotonic (DM) [78], assign priorities to task instances that do not change with time. For example under DM, priorities of instances of periodic tasks are inversely related to their deadline parameter, *i.e.*, a smaller deadline parameter implies higher priority. Although EDF and DM are popular scheduling algorithms used in elementary systems, many other algorithms such as modified RM [99], Audsley's priority assignment [10], etc., have also been developed.

Embedded systems bridge the world of discrete computation and communication with the physical world of sensors, actuators and controllers. They use the powerful world of computing to monitor and control the continuous dynamics of physical and engineered systems. Many of these systems are comprised of time-sensitive functionality, similar to those in elementary real-time systems. However, unlike elementary systems, embedded real-time systems are severely resource

constrained due to size and power limitations. Furthermore, they are frequently interacting with the environment and are providing more and more functionalist's; thus increasing in scale and complexity. Application areas mentioned earlier such as avionics, medical devices, automobile, and factory automation, are in fact examples of such embedded real-time systems.

Complex and large-scale software systems, of which embedded real-time systems is a prominent example, require advanced design and analysis methods. Component-based engineering, which involves compositional system modeling, is widely accepted as a good design strategy for these systems (see [45, 63]). It is founded on the paradigm that to design a complex system, we can first decompose it into simpler and smaller components. Components are designed independently and their functional behaviors are abstracted into interfaces, hiding away internal complexities. To generate the whole system, interfaces are composed with each other taking into consideration effects of mutual interactions. Under this paradigm, all interactions between components are required to occur through interfaces. This provides a clean separation between components and also simplifies system verification. Additionally, interfaces could expose only as much information about components as is necessary for their composition, and thus help protect the intellectual property rights of designers. In general, interfaces can be composed in a piecewise manner wherein a set of interfaces are first composed to generate a new component, and this new component's interface is then composed with other interfaces. The system can therefore be specified as a tree of components (and interfaces), where some subset of interfaces are composed at each level. We refer to these systems as *component-based real-time systems* in this dissertation.

1.2 Hierarchical real-time systems

For component-based real-time systems, designers must also model the real-time characteristics of components, in addition to their functional behaviors. These characteristics can then be used

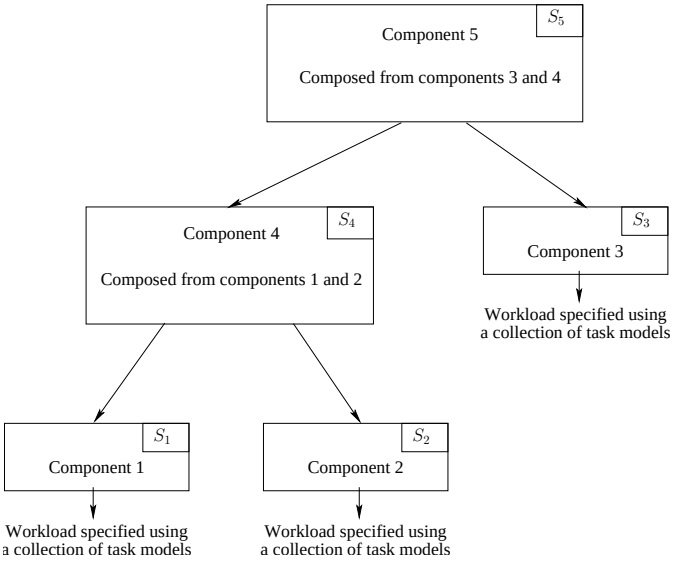


Figure 1.1: Hierarchical real-time system

to determine the overall system resource requirements. One popular technique, based on task models, has been developed for this purpose [47]. In this technique, the workload of an elementary component – component that represents basic functional blocks and is not composed from other components – is specified using task models as in elementary real-time systems. On the other hand, for a component that is composed from interfaces of other components, its workload is specified as a collection of those underlying components. Thus, the real-time characteristics of a component-based system can be represented as a tree of nodes, where each node is defined by its workload and a scheduling algorithm. We call these trees *hierarchical real-time systems*.

A hierarchical real-time system supports hierarchical resource sharing between different components in the system. Using the scheduling algorithm that it employs, a parent node prioritizes and distributes its share of the resource among all its children. This tree of nodes models a scheduling hierarchy, where each node uses its own independent (and possibly different) scheduling algorithm. For example, Figure 1.1 shows a hierarchical real-time system with five components. The root component (component 5) distributes its share of the resource among components 3 and 4. This distribution is based on scheduling algorithm S_5. Similarly, component 4 distributes its

share of the resource among components 1 and 2 under scheduler S_4. Finally, elementary components $1, 2$, and 3, use their share of the resource to schedule tasks in their respective workloads. Scheduling hierarchies are omnipresent in component-based real-time systems such as those found in avionics (design foundation for digital avionics systems, ARINC specification 653-2, part I [55]), virtual machines (Xen hypervisor [41],VMware ESX Server [112]), etc.

For a system with timing constraints, *schedulability* is a fundamental property that is the following question: "How much resource is required to guarantee that all the timing constraints of the system are satisfied?". Since available resources are finite, it is important to develop analysis techniques that address this question in the system design phase itself. For elementary real-time systems, schedulability analysis should compute the amount of resource required to successfully schedule the systems' workload under its scheduler, and such techniques have been developed in the past (see [82, 78, 28, 29, 76, 62, 14, 32]). On the other hand, for hierarchical real-time systems, schedulability analysis should compute the amount of resource required by each component to successfully schedule its workload under its scheduler. Although these problems are similar, hierarchical systems differ from elementary systems in one important aspect and that is the workload. In an elementary system the workload is specified as a collection of task models such as periodic, sporadic, etc. Whereas in a hierarchical system, the workload of a component may be comprised of other components in the system. Therefore, schedulability analysis of hierarchical systems essentially involves handling of such non-trivial workloads.

System designers abstract functional behaviors of components using various modeling techniques like interface automata [44], timed interfaces [46], etc. For hierarchical systems, in addition to functional behaviors, there is a need to abstract extra-functional properties of components in their interfaces. These properties include among others, the amount of resource required from the environment to guarantee that all the deadlines of the component are met. Apart from being

faithful to the component-based design paradigm, abstracting resource requirements of components serves another important purpose explained as follows. When a component's workload comprises of task models like periodic, sporadic, etc., scheduling this workload under popular schedulers like EDF and DM is straightforward as described at the beginning of this section. However, if a component's workload comprises of other components, this is not the case. To schedule this workload under EDF we must abstract it into a sequence of task instances with deadlines, and to schedule it under DM we must abstract it into task models such as periodic, sporadic, etc. Hence, in general, resource requirements of each component in the workload must be abstracted into workload models that the higher level scheduler can schedule.

In summary, real-time characteristics of many component-based systems can be modeled using hierarchical scheduling frameworks. Since these systems operate in highly resource constrained environments, it is desirable to develop techniques that analyze their schedulability in the design phase itself. A fundamental problem that such analysis techniques must address is, "how to abstract the resource requirements of components in their interfaces, so that higher level schedulers can understand and schedule these abstract workloads?". Schedulability of the entire hierarchical system can then be deduced from the abstract workload in the root component's interface. These techniques are called *compositional schedulability analysis techniques*, because they uphold the principles of component-based engineering.

Many studies have been proposed in the past on compositional schedulability analysis techniques [97, 80, 102, 101, 2, 42, 85, 43, 113, 114, 67, 107]. These studies can be broadly classified into two categories based on how they abstract the resource requirements of components. Thiele *et. al.* [113, 114, 107] have developed demand bound function based abstractions by applying network calculus theory into real-time context [106]. Given a workload, scheduling algorithm to prioritize it, and time interval length, demand bound function gives the largest resource requirements of the workload in any time interval of that length. Another approach to abstract resource

requirements of components is using *resource models* or *servers* [97, 80, 102, 101, 2, 42, 85, 43]. A resource model is a model for specifying the characteristics of resource supply. For example, a periodic resource model [102] or server [97, 80] with period Π and resource capacity Θ, specifies a resource supply that provides Θ units of resource in every Π time units. To abstract resource requirements using such models, these studies defined a supply bound function. Given a resource model and time interval length, this function gives the smallest resource supply guaranteed by the resource model in any time interval of that length. They also defined transformations from resource models to well known tasks (*e.g.*, from periodic resource to periodic task), and used these tasks as abstract workloads for higher level schedulers.

1.3 Contributions and organization

This dissertation develops new compositional techniques that improve the state of the art in schedulability analysis of hierarchical real-time systems. For systems scheduled on uniprocessor platforms (hardware platforms with a single processor), we address two fundamental issues; component interface re-usability in dynamic systems, and resource overheads of abstract workloads. We develop new incremental analysis techniques such that interface generation is independent of the order in which elements of a workload are considered. These techniques reuse interfaces to analyze workload modifications, giving rise to efficient on-the-fly admission tests for components. Towards eliminating resource under-utilization in abstract workloads, we first develop a new resource model called *Explicit Deadline Periodic* (EDP), and show that the resulting resource overheads are smaller than those for presently known models such as the periodic resource model. We also characterize the property of *optimal resource utilization* in hierarchical systems, and show that although hard in general, certain restricted notions of optimality can be achieved.

In the latter part of this dissertation, we consider real-time systems scheduled on multiprocessor platforms (hardware platforms with more than one processor). Based on the concept of scheduling hierarchies, we develop new scheduling algorithms and analysis techniques for elementary real-time systems. We show that virtual clustering of workloads, which involves splitting workloads into components and then scheduling them using hierarchical schedulers, increases the resource utilization bound of some existing algorithms like US-EDF$\{m/(2m-1)\}$ [105].

Incremental analysis. Embedded systems often operate in a dynamic setting, where the environment and user requirements are frequently modified. This means that component workloads also undergo frequent changes, indicating a need for analysis techniques that can reuse component interfaces. Interfaces for modified components should be generated using their old interfaces and information on the changes. Such reuse improves the efficiency of various on-line system adaption techniques like dynamic voltage scaling (see [12, 96]) and on-line admission tests (see [114]). However, analysis techniques that reuse interfaces must satisfy an important property called *incrementality*. This property states that the analysis should be independent of the order of abstraction of workload elements. In other words, if the workload of a component can change in multiple ways resulting in the same final workload for each sequence of changes, then the same abstract workload must be generated in the component's interface for each of those sequences. Otherwise, the same component would require different amounts of resource depending on the order in which its workload elements are considered. In Chapter 4 (also see [54, 51]) we use periodic resource model based workload abstractions to develop new incremental analysis techniques that support interface reuse. To compare our technique with existing non-incremental approaches based on the same resource model, we also derive a resource utilization bound.

Optimal resource utilization. The abstract workload in a component's interface represents the amount of resource required to successfully schedule the component's actual workload. It is therefore important to reduce (or if possible eliminate) the resource gap between the abstract

and actual workloads, so that resource requirements of the system as a whole can be reduced.

In Chapter 5 (also see [49]) we introduce a new resource model called Explicit Deadline Periodic (EDP) and develop compositional analysis techniques based on it. EDP models are similar to periodic resource models, except that they have an explicit deadline parameter Δ as opposed to the implicit deadline Π of periodic models. We show that abstract workloads generated from EDP models have smaller resource requirements than those generated from periodic models. This is because the additional deadline parameter enables a tighter control over resource supply.

Although many compositional analysis techniques have been developed for hierarchical systems, there is no work that quantifies optimal resource utilization in such systems. As a result, quantitative assessment of various analysis techniques is not possible on a system level; workload abstractions can be assessed locally at component level. Towards addressing this issue, in Chapter 6 (also [50]), we define two notions of optimal resource utilization for abstract workloads; *load-based* and *demand-based*. The former identifies abstractions whose average resource requirements over time (load) match those of the actual workload. Whereas, the latter identifies abstractions whose exact resource requirements over time (demand) match those of the actual workload. We develop efficient analysis techniques to generate load optimal abstract workloads, and also show that, in general, no realizable resource model based abstraction can achieve this optimality. However, demand optimal abstractions are hard to generate using our techniques (exponential complexity), and we justify this complexity with an example.

In Chapter 7 (also [52]) we present a case-study based on ARINC specification 653-2 [55], which describes the design foundations for digital avionics systems. We analyze avionics workloads to show that our techniques generate efficient workload abstractions in practice.

Multiprocessor scheduling. With the emergence of many commercial multi-cores such as Intel's Xeon, it is believed that the next generation of embedded systems will have the capabilities

for concurrent scheduling. Therefore, scheduling of elementary real-time systems on multiprocessor platforms is an area that has received much attention in the recent past. In general, these scheduling algorithms follow a two step process; assigning tasks to processor clusters in order to regulate task migrations and thereby control concurrency[1], and scheduling tasks on these processor clusters. However, they consider only restricted forms of task-processor mappings; *partitioned* in which tasks are statically assigned to individual processors, and *global* in which each task is allowed to execute on any processor in the platform. As a result, very few optimal scheduling algorithms[2] for periodic and sporadic task systems have been developed so far.

In Chapter 8 (also [100]) we propose another, more general, approach for task-processor mappings called *virtual cluster-based scheduling*. In this approach, each task is statically assigned to a virtual processor cluster, tasks in each cluster are globally scheduled among themselves, and clusters in turn are globally scheduled on the multiprocessor platform. This two level cluster-based scheduling paradigm (intra- and inter-cluster scheduling) can also be viewed as a two level hierarchical real-time system. Therefore, we develop abstract workload based techniques to support virtual clustering, taking into consideration properties that minimize the resource utilization of these abstractions. This virtualization has the ability to redistribute unused resource from one cluster to another. Using this flexibility we develop a new optimal scheduling algorithm for sporadic task systems in which every task has its deadline equal to its period, and also improve the resource utilization bound of US-EDF$\{m/(2m-1)\}$ scheduler [105].

Prior to describing these techniques, we discuss related work in Chapter 3, and introduce notations, scheduling algorithms, and existing compositional analysis techniques in Chapter 2.

[1] Maximizing concurrency leads to delaying of idle processor slots, which in turn increases resource utilization.

[2] Informally, a scheduling algorithm is optimal for a certain type of workload, if it can successfully schedule every workload of that type that is schedulable under some scheduling algorithm.

Chapter 2

System descriptions and background

In this chapter we introduce workload models, hardware platforms, scheduling algorithms, hierarchical system notations, and resource models, used in this dissertation. We also describe existing compositional analysis techniques based on the periodic resource model [102].

2.1 System descriptions and notations

In discrete computing systems, it is always possible to identify a time interval such that no successive events or actions can occur within this interval (*e.g.*, system clock frequency can be used to define this smallest time unit). We assume that all the notations used in this dissertation are defined in terms of this time unit.

2.1.1 Workload models

Each instance of resource demand is called a *real-time job* and it can be specified using three parameters; release instant r with respect to an origin of time, maximum required resource capacity c, and relative deadline d with respect to the release instant. This job requires c units of resource in the time interval $(\mathrm{r}, \mathrm{r}+\mathrm{d}]$.

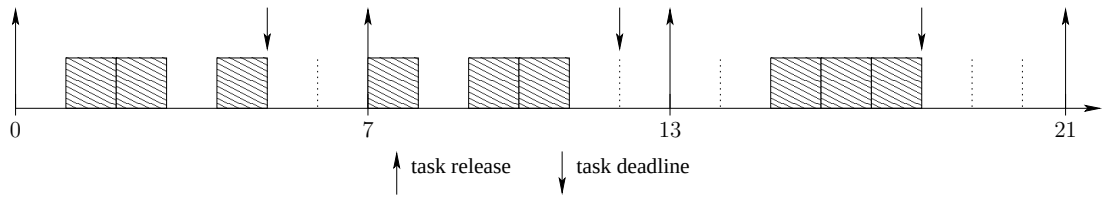


Figure 2.1: Release and execution pattern for sporadic task $\tau = (6, 3, 5)$

Workloads of elementary real-time systems are often described using infinite sets of jobs (recurring task models), of which periodically recurring tasks [82], and aperiodically or sporadically recurring tasks [88] are the most common. A periodic task $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$ has release separation T, maximum resource capacity requirement C, and relative deadline D, where $\mathrm{C} \leq \mathrm{D}$ and $\mathrm{C} \leq \mathrm{T}$. Each job of this task requires C units of resource within D time units from its release. Also, two successive jobs are released with a separation of T time units. A sporadic task is similar to a periodic task, except that T now denotes the minimum separation between successive job releases as opposed to exact separation. For example, Figure 2.1 shows a sample release and execution pattern of sporadic task $\tau = (6, 3, 5)$. A task is denoted as *constrained deadline* if $\mathrm{D} < \mathrm{T}$, *implicit deadline* if $\mathrm{D} = \mathrm{T}$, and *arbitrary deadline* if $\mathrm{D} \neq \mathrm{T}$. From these descriptions it is easy to deduce that periodic tasks are used for modeling time-triggered actions, whereas sporadic tasks are used for modeling event-triggered actions. Together, these tasks can model resource demands in many component-based systems, and hence they are extensively used (see [82, 88, 29]).

In this dissertation we use periodic and sporadic task models to represent the workloads of elementary real-time systems. We also use these models to represent the workloads of elementary components in hierarchical systems. Each workload is represented by a finite set of periodic and sporadic tasks. The workload is denoted as implicit deadline if every task in the workload is implicit deadline, constrained deadline if every task is constrained or implicit deadline, and arbitrary deadline otherwise.

2.1.2 Hardware platforms

Uniprocessor platforms. In the initial parts of this dissertation (Chapters 4, 5, 6, and 7) we consider uniprocessor hardware platforms for scheduling hierarchical systems. Each platform comprises of one processor that executes with unit bandwidth – t units of processor capacity in any time interval of length t. Voltage scaling techniques (*e.g.* [12, 96]), which are extremely popular in power constrained embedded systems, may reduce the available bandwidth of processors on these platforms. Although this dissertation only considers uniprocessors having bandwidth one, the analysis techniques presented here can be trivially extended for such low bandwidth processors.

Multiprocessor platforms. In the latter part of this dissertation (Chapter 8) we consider multiprocessor platforms for scheduling elementary real-time systems. Each platform comprises of m processors and each processor executes with unit bandwidth. Such a platform is called identical, unit-capacity multiprocessor platform.

In this dissertation, we do not consider other platform resources like memory and network links in schedulability analysis. Hence, we use the general term resource to refer to processors.

2.1.3 Scheduling algorithms

On uniprocessor platforms, we mainly consider two scheduling algorithms; dynamic-priority Earliest Deadline First (EDF) and fixed-priority Deadline Monotonic (DM). In *dynamic-priority* algorithms, the relative priority ordering of jobs of two tasks vary over time. Suppose J_1 and J_2 represent two jobs of periodic or sporadic task τ_1, and J_3 and J_4 represent two jobs of periodic or sporadic task τ_2. Then, in dynamic-priority algorithms, it is possible that J_1 has higher priority than J_3, but J_2 has lower priority than J_4. On the other hand, in *fixed-priority* algorithms, the relative priority ordering of jobs of two tasks is preserved over time. In the above example, if J_1 has higher priority than J_3, then J_2 will also have higher priority than J_4.

Consider a periodic or sporadic task set $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$. At

each time instant t, EDF schedules a unfinished job that has the earliest relative deadline. We assume that ties are broken in favor of smaller task indices.

Definition 2.1 (EDF scheduling) *Let $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ denote a task set. Let t denote the current time instant and $\mathcal{J} = \{J_1 = (\mathrm{r}_1, \mathrm{c}_1, \mathrm{d}_1), \ldots, J_n = (\mathrm{r}_n, \mathrm{c}_n, \mathrm{d}_n)\}$ denote the set of jobs at t, such that each J_i is a job of task τ_i. If some task τ_j has no unfinished job at t, then let $\mathrm{c}_j = 0$. In the interval $[t, t+1)$,* EDF *will schedule a job J_i such that (1) $\mathrm{c}_i > 0$, (2) $\mathrm{r}_i + \mathrm{d}_i \leq \mathrm{r}_k + \mathrm{d}_k$ for all $k \neq i$, and (3) $i < k$ for any k which satisfies the condition $\mathrm{r}_i + \mathrm{d}_i = \mathrm{r}_k + \mathrm{d}_k$.*

Figure 2.2 shows the EDF schedule of a task set $\{\tau_1 = (4, 2, 4), \tau_2 = (7, 3, 7)\}$. The following theorem records an important property of EDF scheduler.

Theorem 2.1 (Also in [48]) EDF *is an optimal uniprocessor scheduling algorithm for arbitrary deadline periodic and sporadic task sets,* i.e.*, if any scheduling algorithm can schedule such a task set on a uniprocessor platform, then* EDF *can also schedule it on the uniprocessor platform.*

Under DM, jobs are prioritized based on the deadline parameter of the corresponding task specification; at each time instant, DM schedules a unfinished job with smallest value for D. We assume that ties are broken in favor of smaller task indices.

Definition 2.2 (DM scheduling) *Let $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ denote a task set. Let t denote the current time instant and $\mathcal{J} = \{J_1 = (\mathrm{r}_1, \mathrm{c}_1, \mathrm{d}_1), \ldots, J_n = (\mathrm{r}_n, \mathrm{c}_n, \mathrm{d}_n)\}$ denote the set of jobs at t, such that each J_i is a job of task τ_i. If some task τ_j has no unfinished job at t, then let $\mathrm{c}_j = 0$. In the interval $[t, t+1)$,* DM *will schedule a job J_i such that (1) $\mathrm{c}_i > 0$, (2) $\mathrm{D}_i \leq \mathrm{D}_k$ for all $k \neq i$, and (3) $i < k$ for any k satisfying condition $\mathrm{D}_i = \mathrm{D}_k$.*

Figure 2.2 also shows the DM schedule of task set $\{\tau_1 = (4, 2, 4), \tau_2 = (7, 3, 7)\}$. The following theorem records an important property of DM schedulers.

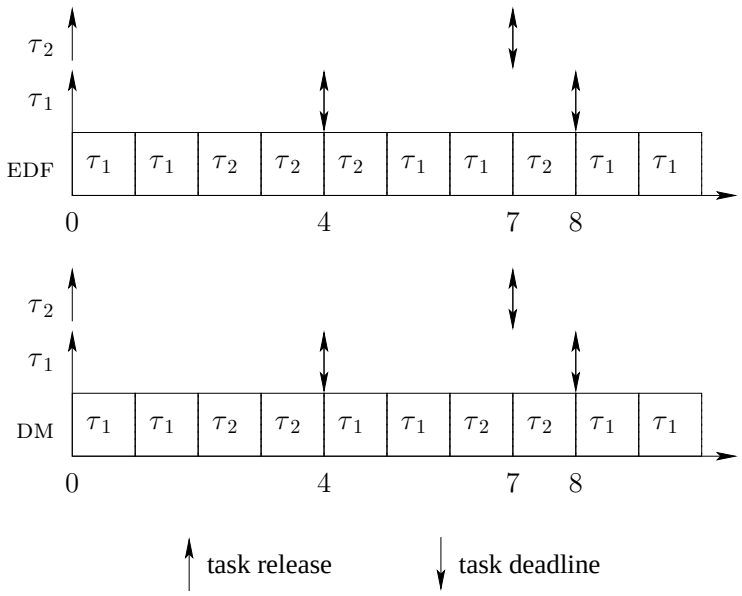


Figure 2.2: EDF and DM schedules for task set $\{\tau_1 = (4, 2, 4), \tau_2 = (7, 3, 7)\}$

Theorem 2.2 (Also in [78]) DM *is an optimal uniprocessor fixed-priority scheduling algorithm for constrained deadline periodic and sporadic task sets,* i.e.*, if any fixed-priority scheduling algorithm can schedule such a task set on a uniprocessor platform, then* DM *can also schedule it on the uniprocessor platform.*

Hence, although under EDF we do not restrict the task deadlines in any manner, we assume that the workload is constrained deadline under DM.

On multiprocessor platforms, we consider dynamic-priority global EDF scheduling algorithm (denoted as gEDF), which reduces to EDF on uniprocessors. At each time instant, gEDF schedules unfinished jobs that have the m earliest relative deadlines, where m denotes number of processors on the platform. If there are fewer than m unfinished jobs, then it schedules all of them.

Definition 2.3 (gEDF scheduling) *Let* $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ *denote a task set. Let* t *denote the current time instant and* $\mathcal{J} = \{J_1 = (\mathrm{r}_1, \mathrm{c}_1, \mathrm{d}_1), \ldots, J_n = (\mathrm{r}_n, \mathrm{c}_n, \mathrm{d}_n)\}$ *denote the set of jobs at* t*, such that each* J_i *is a job of task* τ_i*. If some task* τ_j *has*

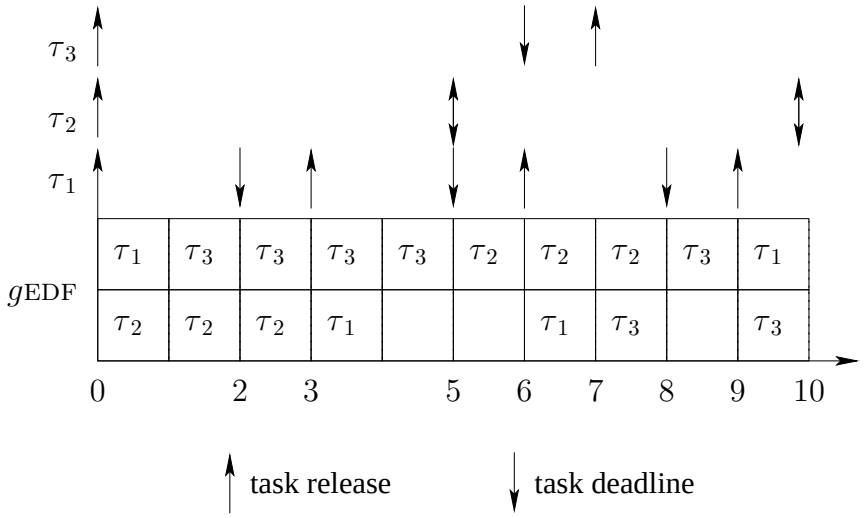


Figure 2.3: gEDF schedule for $\{\tau_1 = (3, 1, 2), \tau_2 = (5, 3, 5), \tau_3 = (7, 4, 6)\}$

*no unfinished job at t, then let $\mathrm{c}_j = 0$. Also, let m denote the number of processors on the multiprocessor platform. If number of elements in $\mathcal{J}$ with $\mathrm{c} > 0$ is at most m, then g*EDF *will schedule all jobs in $\mathcal{J}$ in the interval $[t, t+1)$. Otherwise, suppose $\mathcal{J}$ is sorted such that, if J_{k_i} occurs before J_{k_j} in the sorted list, then (1) $\mathrm{r}_{k_i} + \mathrm{d}_{k_i} \leq \mathrm{r}_{k_j} + \mathrm{d}_{k_j}$, and (2) $k_i < k_j$ whenever $\mathrm{r}_{k_i} + \mathrm{d}_{k_i} = \mathrm{r}_{k_j} + \mathrm{d}_{k_j}$. Suppose $J_{k_1}, \ldots, J_{k_m}$ denotes the first m elements in this sorted list, after ignoring all jobs with $\mathrm{c} = 0$. Then, g*EDF *will schedule $J_{k_1}, \ldots, J_{k_m}$ in the interval $[t, t+1)$.*

Figure 2.3 shows the gEDF schedule of task set $\{\tau_1 = (3, 1, 2), \tau_2 = (5, 3, 5), \tau_3 = (7, 4, 6)\}$ on a platform comprising of two processors. Among other things, global schedulers are work-conserving (no processor is idle when there is back-logged work), and typically result in lower average number of preemptions when compared to partitioned schedulers (see [31]).

2.1.4 Component model

A component in a hierarchical system is comprised of a real-time workload and a scheduling policy for the workload. We model workloads of elementary components using independent periodic and sporadic task sets. Also, we restrict component scheduling algorithms to EDF and DM.

Definition 2.4 (Elementary real-time component) *An* elementary real-time component $\mathcal{C}$

is specified as $\mathcal{C} = \langle \mathcal{W}, \mathcal{S} \rangle$*, where* $\mathcal{W} (= \mathcal{T})$ *is a finite set of independent periodic and sporadic tasks, and* $\mathcal{S}$ *(=* EDF *or* DM*) is the scheduling policy.*

A non-elementary component can then be defined as follows.

Definition 2.5 (Non-elementary real-time component) *A* non-elementary real-time component $\mathcal{C}$ *is specified as* $\mathcal{C} = \langle \mathcal{W}, \mathcal{S} \rangle$*, where* $\mathcal{S} =$ EDF/DM *and* $\mathcal{W} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\} \bigcup \mathcal{T}$*. Each* $\mathcal{C}_i$ *is a real-time component (elementary or non-elementary) and* $\mathcal{T}$ *is a finite set of independent periodic and sporadic tasks.*

This component structure can be viewed as a directed graph with components or task sets as nodes of the graph. Whenever $\mathcal{W}$ is given by Definition 2.4, there is an edge from a node denoting $\mathcal{C}$ to a node denoting $\mathcal{T}$. Similarly, whenever $\mathcal{W}$ is given by Definition 2.5, in addition to the edge from $\mathcal{C}$ to $\mathcal{T}$, there is an edge from $\mathcal{C}$ to each $\mathcal{C}_i$. A *hierarchical real-time system* is then specified as $\mathcal{H} = \langle \mathcal{C}, \mathcal{S} \rangle$, where $\mathcal{C}$ denotes a real-time component whose underlying undirected graph is a tree, and $\mathcal{S}$ denotes the scheduler under which $\mathcal{C}$ is scheduled on the hardware platform. $\mathcal{C}$ is also called the *root component* of system $\mathcal{H}$. Figure 2.4 shows a hierarchical system in which $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ are elementary components, and $\mathcal{C}_4$ and $\mathcal{C}_5$ are non-elementary components.

2.1.5 Schedulability analysis

The property of *schedulability*, as discussed in Section 1.2, refers to the amount of resource required to satisfy all the timing constraints in a system. Given an environment under which the system operates (resource supply from hardware platform), *schedulability analysis* refers to the process of "determining whether the amount resource supplied by the environment is sufficient to guarantee schedulability of the system". Since the designer cannot always control the operating environment, it is essential to perform this analysis under adversarial assumptions, *i.e.*,

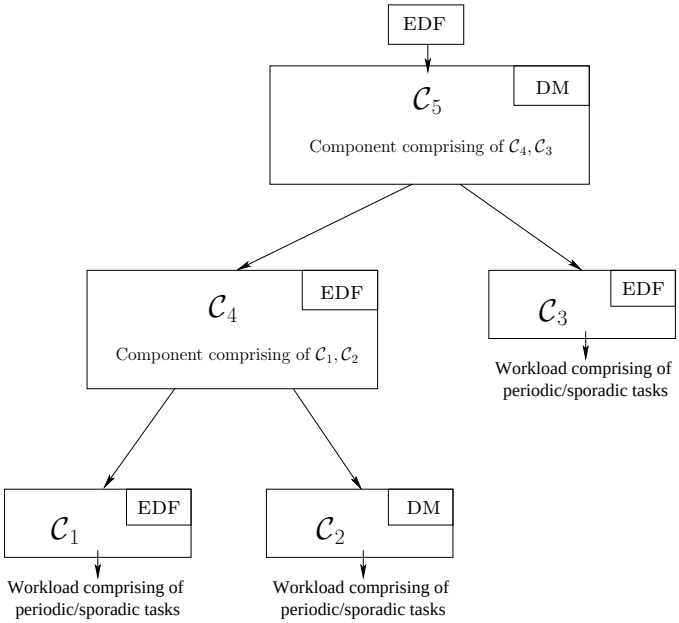


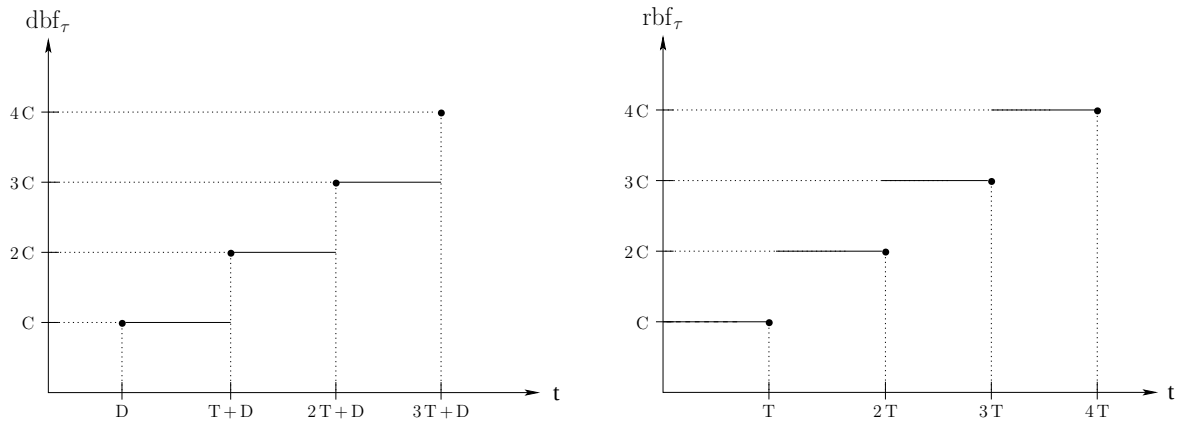
Figure 2.4: Example hierarchical real-time system

considering worst-case (maximum possible) resource demand of the system and worst-case (minimum guaranteed) resource supply from the environment. Furthermore, closed-form expressions for worst-case resource demand and supply are required to enable design-time schedulability analysis. In the following two sections, we present these closed-form expressions for component-based real-time systems scheduled on a uniprocessor platform.

Component resource demand

The resource demand of a periodic or sporadic task is generated by an infinite sequence of real-time jobs. Although the release pattern for jobs of periodic tasks is fixed, this is not the case for sporadics. To determine the worst-case resource demand for schedulability analysis, it is then essential to identify the corresponding worst-case job release pattern. For sporadic tasks, this occurs when successive jobs are released as soon as possible, *i.e.*, with release separation T for a sporadic task $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$. It is trivially true that this pattern generates the maximum possible (worst-case) resource demand in any time interval.

The *demand bound function* (dbf : $\Re \rightarrow \Re$) of a periodic or sporadic task, introduced by



(a) Demand bound function (b) Request bound function

Figure 2.5: Demand and request bound functions of task $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$

Baruah *et. al.* [28], gives a closed-form expression for the worst-case resource demand of the task. For a real number t, $\mathrm{dbf}(t)$ gives the maximum possible resource requirements of the task in any time interval of length t. While computing this demand, dbf only uses the resource requirements of jobs which are released and have their deadline within the time interval under consideration. Then, it is easy to observe that this worst-case demand occurs for those intervals whose start time coincides with the release time of any job. Definition 2.6 gives the dbf for a periodic or sporadic task, and the function itself is plotted in Figure 2.5(a).

Definition 2.6 (Demand bound function (task)) *The demand bound function of a periodic or sporadic task* $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$ *is specified as*

$$\mathrm{dbf}_\tau(t) = \max\left\{0, \left\lfloor \frac{t + \mathrm{T} - \mathrm{D}}{\mathrm{T}} \right\rfloor\right\} \mathrm{C}$$

The restriction that dbf only uses jobs whose release time and deadline are both in the time interval under consideration seems counter-intuitive for determining the worst-case demand. For example, consider the task $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$ whose dbf is plotted in Figure 2.5(a). Suppose $\mathrm{D} = \mathrm{C}$ and $t = \epsilon$, where $0 < \epsilon < \mathrm{D}$. Although τ requires at least t units of resource in any time interval of length t, $\mathrm{dbf}_\tau(t) = 0$. We now describe a property of the uniprocessor platform on which

τ is scheduled, that allows for this counter-intuitive definition of dbf. Since dbf(D) = C, the uniprocessor must provide at least C resource units in any time interval of length D, to guarantee schedulability of task τ. Now, since the uniprocessor has bandwidth one, it can provide at most ϵ resource units in a time interval of length ϵ. This means that in order to provide C resource units in any time interval of length D (= C), it must provide at least ϵ resource units in any time interval of length ϵ. In other words, analyzing the schedulability at t = D also guarantees schedulability for all interval lengths $t <$ D. Since we use dbf only for schedulability analysis, Definition 2.6 of dbf is sufficient for our purposes.

Although dbf presents a useful demand bound under EDF, it cannot be used to analyze schedulability of tasks under DM. This is because, under DM, job priorities are determined by the deadline parameter of corresponding tasks, instead of actual job deadlines. As a result, all jobs released in a time interval can contribute to the resource demand in that interval under DM, irrespective of whether their deadlines lie in the interval or not. Hence, Lehoczky *et. al.* [76], defined another demand function called the *request bound function* (rbf : $\Re \rightarrow \Re$). For a real number t, rbf(t) gives the maximum possible resource requested by jobs of the task in any time interval of length t. In this case as well, it is easy to observe that the worst-case resource request occurs for those intervals whose start time coincides with the release time of any job. Definition 2.7 gives the rbf for a periodic or sporadic task, and the function itself is plotted in Figure 2.5(b).

Definition 2.7 (Request bound function (task)) *The request bound function of a periodic or sporadic task* $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$ *is specified as*

$$\mathrm{rbf}_\tau(t) = \left\lceil \frac{t}{\mathrm{T}} \right\rceil \mathrm{C}$$

Consider an elementary component that uses the EDF scheduler. The demand bound function for this component, introduced by Baruah *et. al.* [28], specifies the worst-case resource demand

of all the tasks in the component. This function only considers those jobs whose release time and deadline are both in the time interval under consideration. It has been shown that this worst-case occurs in a time interval in which (1) the first job of all the tasks are released synchronously at the beginning of the interval, and (2) every successive job of each task is released as soon as possible [28]. The component demand bound function can be defined as follows.

Definition 2.8 (Demand bound function (component)) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of periodic or sporadic tasks. The demand bound function of component $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$ considering demand only from tasks $\{\tau_1, \ldots, \tau_i\}$, with $1 \leq i \leq n$, is given by*

$$\text{dbf}_{\mathcal{C},i} = \sum_{j=1}^{i} \text{dbf}_{\tau_j}$$

In particular, we define $\text{dbf}_{\mathcal{C}} = \text{dbf}_{\mathcal{C},n}$.

Similarly, consider an elementary component that uses the DM scheduler. The request bound function for this component, introduced by Lehoczky *et. al.* [76], specifies the maximum possible resource requested by all the tasks in the component. This function considers those jobs whose release occur in the time interval under consideration, irrespective of whether their deadlines lie in the interval or not. It has been shown that this worst-case occurs in a time interval with properties identical to those described in the previous paragraph [76]. Using rbf_{τ} from Definition 2.7, the component request bound function can be defined as follows.

Definition 2.9 (Request bound function (component)) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of periodic or sporadic tasks. The request bound function of component $\mathcal{C} = \langle \mathcal{T}, \text{DM} \rangle$ considering resource requests only from tasks $\{\tau_1, \ldots, \tau_i\}$, with $1 \leq i \leq n$, is given by*

$$\text{rbf}_{\mathcal{C},i} = \sum_{j=1}^{i} \text{rbf}_{\tau_j}$$

Scheduling semantics in hierarchical systems

In this section, we present conditions for schedulability analysis of elementary components scheduled on uniprocessor platforms. For this purpose, we make use of the worst-case resource demand (request) functions presented in the previous section. The following theorem gives the schedulability condition for an elementary component $\mathcal{C}$ scheduled on a uniprocessor under EDF.

Theorem 2.3 (Schedulability under EDF (uniprocessor) [28]) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of periodic or sporadic tasks. Component $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$ is schedulable on a uniprocessor iff*

$$\forall t \ s.t. \ 0 < t \leq L, \text{dbf}_{\mathcal{C}}(t) \leq t, \tag{2.1}$$

where $L = \min\left\{\text{LCM} + \max_{i=1,\ldots,n} \text{D}_i, \frac{U_{\mathcal{T}}(\max_{i=1,\ldots,n}(\text{T}_i - \text{D}_i))}{1-U_{\mathcal{T}}}\right\}$, LCM being the least common multiple of $\text{T}_1, \ldots, \text{T}_n$ and $U_{\mathcal{T}} = \sum_{i=1}^{n} \frac{\text{C}_i}{\text{T}_i}$.

In Equation (2.1), $\text{dbf}_{\mathcal{C}}(t)$ in LHS denotes the worst-case resource demand of component $\mathcal{C}$, and t in RHS denotes the minimum guaranteed (worst-case) resource supply of the uniprocessor platform in any time interval of length t. Since this equation asserts that the maximum possible demand is at most the minimum guaranteed supply, it ensures schedulability of component $\mathcal{C}$ under all combinations of actual resource demand and supply. The *schedulability load* of component $\mathcal{C}$ is defined as $\max_{t \in (0,L]} \frac{\text{dbf}_{\mathcal{C}}(t)}{t}$ and denoted as $\text{LOAD}_{\mathcal{C}}$. If a uniprocessor provides $t \times \text{LOAD}_{\mathcal{C}}$ units of resource in any time interval of length t, then it can successfully schedule component $\mathcal{C}$. Note that since EDF is an optimal scheduler for periodic and sporadic tasks, the *feasibility load* of task set $\mathcal{T}$ ($\text{LOAD}_{\mathcal{T}}$) is also equal to $\text{LOAD}_{\mathcal{C}}$. This means that if a uniprocessor has bandwidth smaller than $\text{LOAD}_{\mathcal{C}}$, then it cannot successfully schedule $\mathcal{T}$ under any scheduling algorithm. Also, $U_{\mathcal{T}}$ in the above definition denotes the total utilization of task set $\mathcal{T}$.

The following theorem gives the schedulability condition for an elementary component $\mathcal{C}$ scheduled on a uniprocessor under DM.

Theorem 2.4 (Schedulability under DM (uniprocessor) [76]) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of periodic or sporadic tasks. Component $\mathcal{C} = \langle \mathcal{T}, \text{DM} \rangle$ is schedulable on a uniprocessor iff*

$$\forall i, \exists t \in [0, \mathrm{D}_i] \ s.t. \ \ \mathrm{rbf}_{\mathcal{C},i}(t) \leq t \tag{2.2}$$

The *schedulability load* of component $\mathcal{C}$ is defined as $\max_{i=1,\ldots,n} \min_{t \in (0,\mathrm{D}_i]} \frac{\mathrm{rbf}_{(\mathcal{C},i)}(t)}{t}$.

For elementary components, scheduling its workload (periodic/sporadic task set) under EDF or DM is straightforward as explained in Section 2.1.3. However, if a component $\mathcal{C}$'s workload comprises of other components, this is not the case. To schedule this workload under EDF we must present a set of real-time jobs, and to schedule it under DM we must present a task set like periodic, sporadic, etc. In other words, each component $\mathcal{C}_i$ in the workload of $\mathcal{C}$ must be transformed into a set of tasks/jobs that $\mathcal{C}$'s scheduler can schedule. Furthermore, informally, these tasks/jobs should be such that the amount of resource required by them under $\mathcal{C}$'s scheduler, must be at least as much as the amount of resource required by component $\mathcal{C}_i$. We refer to these tasks/jobs as $\mathcal{C}_i$'s *interface*.

Definition 2.10 (Component interface) *We define the interface of a task set to be the task set itself. Consider a component $\mathcal{C} = \langle \mathcal{W}, \mathcal{S} \rangle$ with $\mathcal{W} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$. Let $\mathcal{C}$ itself be scheduled under $\mathcal{S}'$, and $\mathcal{I}_\mathcal{W}$ denote the set of interfaces of workload $\mathcal{W}$. $\mathcal{I}_\mathcal{C}$ is an interface for $\mathcal{C}$ iff $\mathcal{I}_\mathcal{C}$ is schedulable under $\mathcal{S}'$ implies $\mathcal{I}_\mathcal{W}$ is schedulable under $\mathcal{S}$, where schedulability is determined using Theorems 2.3 and 2.4. In this definition, we assume that $\mathcal{I}_\mathcal{W}$ executes under $\mathcal{S}$ whenever $\mathcal{I}_\mathcal{C}$ is scheduled by $\mathcal{S}'$.*

Thus, given a component $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \mathcal{S} \rangle$, the fundamental question in scheduling $\mathcal{C}$ is, "What is the interface that each $\mathcal{C}_i$ must present to $\mathcal{S}$?". Since we use Theorems 2.3 and 2.4 to define interface schedulability in the above definition, we require that interfaces be represented

using periodic or sporadic task sets. Resource models, described in this dissertation, can be used to represent component interfaces, provided transformations from such models to periodic or sporadic tasks are also developed. Throughout the dissertation, we will utilize this flexibility in interface representation to simplify presentation of our work.

2.2 Resource model based compositional analysis

In this section we describe previous work on compositional schedulability analysis of hierarchical systems. We focus on techniques that abstract component workloads using resource models [97, 80, 102, 101, 2, 42, 85, 43], and in particular, we describe in detail analysis based on the periodic resource model(see [97, 80, 102]).

2.2.1 Resource model

To satisfy the resource requirements of a real-time task or component, the hardware platform must supply sufficient processor capacity. A *resource model* is a model for specifying the timing properties of this resource supply. Two resource models, periodic [97, 102] and bounded-delay[1] [90], have been proposed in the past, and in this section we focus on the former. This choice is motivated by the fact that periodic resource models can be trivially transformed into periodic tasks, whereas no such transformation is known for bounded delay models. As a result, periodic models are better suited for generating component interfaces. We now formally define periodic resource models.

Definition 2.11 (Periodic resource model) *A periodic resource model that periodically provides* Θ *units of resource with period* Π *is specified as* $\phi = \langle \Pi, \Theta \rangle$. *For this model* $\frac{\Theta}{\Pi}$ *denotes the* resource bandwidth.

ϕ guarantees a minimum resource supply of Θ units in every successive period of Π time units. Furthermore, the model is flexible in that this resource supply can be provided in an arbitrary

[1] Bounded-delay resource models are described in Section 3.1.1 in Chapter 3.

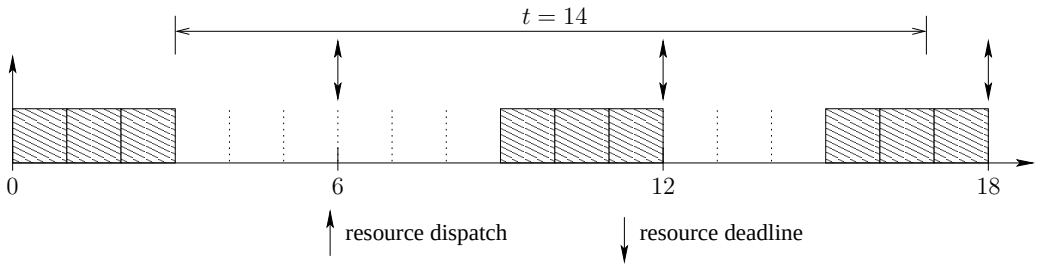


Figure 2.6: Resource supply of model $\phi = \langle 6, 3 \rangle$ corresponding to $\text{sbf}_\phi(14)$

fashion within any single period. For schedulability analysis of a component scheduled using a periodic resource model, closed form expressions that characterize the minimum guaranteed (worst-case) supply from the resource model must be developed. The supply bound function $\text{sbf} : \Re \rightarrow \Re$, introduced by Shin and Lee [102] and Lipari and Bini [80], serves this purpose. Given a resource model R and real number t, $\text{sbf}_R(t)$ gives the minimum amount of resource that R is guaranteed to supply in any time interval of length t. sbf of ϕ is defined as follows.

Definition 2.12 (Supply bound function (periodic)) *Given a periodic resource model* $\phi = \langle \Pi, \Theta \rangle$*, its supply bound function is specified as*

$$\text{sbf}_\phi(t) = \begin{cases} \left\lfloor \frac{t-(\Pi-\Theta)}{\Pi} \right\rfloor \Theta + \max\left\{0, t - 2(\Pi-\Theta) - \left\lfloor \frac{t-(\Pi-\Theta)}{\Pi} \right\rfloor \Pi\right\} & t \geq \Pi - \Theta \\ 0 & \textit{Otherwise} \end{cases} \tag{2.3}$$

In the schedulability analysis conditions that have been developed, a linear lower bound of sbf *is often used to reduce the complexity of analysis. We therefore present this lower bound below.*

$$\text{lsbf}_\phi(t) = \frac{\Theta}{\Pi}\left[t - 2\left(\Pi - \Theta\right)\right] \tag{2.4}$$

Figure 2.7 shows the supply bound function of periodic model ϕ. The largest time interval with no supply (henceforth denoted as *blackout interval length*) is $2(\Pi - \Theta)$. Thereafter the model provides Θ units of resource in every Π time units. Also, as shown in the figure, lsbf_ϕ is a tight lower bound of sbf_ϕ. Corresponding to sbf value for each interval length, there exists a resource

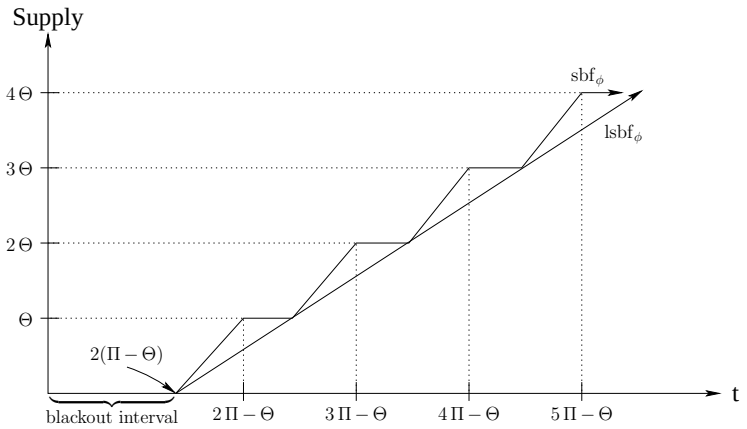


Figure 2.7: Supply bound function of $\phi = \langle \Pi, \Theta \rangle$

supply pattern which generates that value. For example, the resource supply pattern of a periodic model $\phi = \langle 6, 3 \rangle$ corresponding to $\text{sbf}_\phi(14)$ is shown in Figure 2.6. Resource supply in the first period (interval $(0, 6]$) is assumed to be given as soon as possible, and all successive supplies are assumed to be given as late as possible.

2.2.2 Schedulability conditions

To generate component interfaces, previous studies have used periodic resource models to abstract resource requirements of component workloads [80, 102]. They have defined schedulability conditions using dbf or rbf of components and sbf of periodic models. If the timing constraints in a component workload are met when resource requirements are largest (dbf, rbf) and resource supply is smallest (sbf), then the resource model can always schedule the component's workload.

Theorems 2.5 and 2.6 give conditions for schedulability of an elementary component $\mathcal{C}$ under periodic model ϕ, when $\mathcal{C}$ uses EDF or DM respectively. It is worth noting that these conditions are derived by extending the schedulability conditions in Theorems 2.3 and 2.4 with sbf_ϕ. The following theorems also define the notion of *exactly schedulable*, which identifies conditions when resource models use minimum resource bandwidth to schedule components.

Theorem 2.5 (Schedulability under EDF (periodic resource) [102]) *Consider task set*

$\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, *where for each* i, $\tau_i = (\mathrm{T}_i, \mathrm{C}_i, \mathrm{D}_i)$ *is a periodic or sporadic task. Also, let* LCM *denote the least common multiple of minimum separations* $\mathrm{T}_1, \ldots, \mathrm{T}_n$. *Component* $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$ *is schedulable using a periodic resource model* $\phi = \langle \Pi, \Theta \rangle$ *iff* $\frac{\Theta}{\Pi} \geq U_{\mathcal{T}}$ *and*

$$\forall t \text{ s.t. } 0 < t \leq L, \mathrm{dbf}_{\mathcal{C}}(t) \leq \mathrm{sbf}_{\phi}(t), \tag{2.5}$$

where $L = \min\left\{\mathrm{LCM} + \max_{i=1,\ldots,n} \mathrm{D}_i, \frac{U_{\mathcal{T}}(\max_{i=1,\ldots,n}(\mathrm{T}_i - \mathrm{D}_i))}{1-U_{\mathcal{T}}}\right\}$. *Furthermore,* $\mathcal{C}$ *is* exactly schedulable *by* ϕ *iff in addition to the above condition,* $\exists t$ *s.t.* $\min_{i=1,\ldots,n} \mathrm{D}_i \leq t \leq L$ *and* $\mathrm{dbf}_{\mathcal{C}}(t) = \mathrm{sbf}_{\phi}(t)$.

Theorem 2.6 (Schedulability under DM (periodic resource) [80]) *Consider constrained deadline task set* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, *where for each* i, $\tau_i = (\mathrm{T}_i, \mathrm{C}_i, \mathrm{D}_i)$ *is a periodic or sporadic task. Also, for all* $i < j$, *let* $\mathrm{D}_i \leq \mathrm{D}_j$. *Component* $\mathcal{C} = \langle \mathcal{T}, \text{DM} \rangle$ *is schedulable using a periodic resource model* $\phi = \langle \Pi, \Theta \rangle$ *iff*

$$\forall i : 1 \leq i \leq n, \exists t_i \in [0, \mathrm{D}_i] \text{ s.t. } \mathrm{rbf}_{\mathcal{C},i}(t_i) \leq \mathrm{sbf}_{\phi}(t_i) \tag{2.6}$$

Furthermore, $\mathcal{C}$ *is* exactly schedulable *by* ϕ *iff, in addition to the above condition,* $\forall i : 1 \leq i \leq n, \forall t \in [0, \mathrm{D}_i], \mathrm{rbf}_{\mathcal{C},i}(t) \geq \mathrm{sbf}_{\phi}(t)$.

Since $\mathrm{dbf}_{\mathcal{C}}$ can be computed in $\mathcal{O}(n)$ time for each interval length t, the schedulability condition in Theorem 2.5 can be evaluated in $\mathcal{O}(nL)$ time (there are finite number of time instants in interval $(0, L]$ at which $\mathrm{dbf}_{\mathcal{C}}$ changes). Similarly, since $\mathrm{rbf}_{\mathcal{C},i}$ can also be computed in $\mathcal{O}(i)$ time for each t, the schedulability condition in Theorem 2.6 can be evaluated in $\mathcal{O}(\sum_{i=1}^{n} i\,\mathrm{D}_i) = \mathcal{O}(n^2\,\mathrm{D}_n)$ time. Thus, schedulability checking, under both EDF and DM, has pseudo-polynomial complexity.

2.2.3 Compositional analysis

Previous studies have used periodic resource models in component interfaces by transforming them into periodic tasks [80, 102]. They transform a periodic model $\phi = \langle \Pi, \Theta \rangle$ into the periodic

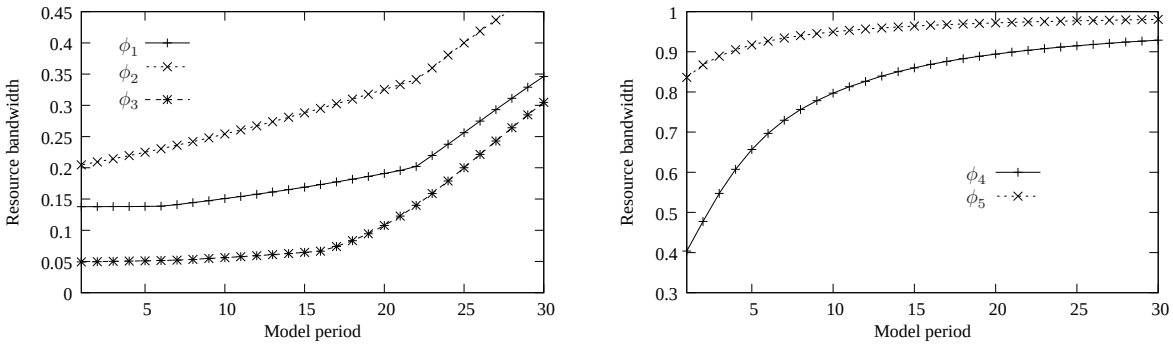
task $\tau_\phi = (\Pi, \Theta, \Pi)$. It is easy to see that the minimum resource requirements of τ_ϕ is at least as much as sbf_ϕ, and therefore this is a valid transformation.

Interfaces for elementary components can be generated using Theorems 2.5 and 2.6. Typically, period values of resource models in these interfaces are fixed a priori by system designers. For instance, one could specify these values taking into consideration preemption related overheads [2]. Compositional schedulability analysis of hierarchical systems can then be achieved as follows. (1) Abstract each elementary component's workload into a periodic resource model $\phi = \langle \Pi, \Theta \rangle$ by computing Θ from Theorem 2.5 or 2.6. To minimize the bandwidth of ϕ, compute the smallest value of Θ that satisfies these theorems (use notion of exactly schedulable). (2) Transform each resource model ϕ into the periodic task τ_ϕ as described above. (3) Repeat the above two steps for components whose workloads are now represented by periodic and sporadic tasks. To reduce the complexity of this analysis, previous studies replaced sbf in these theorems with its linear lower bound (lsbf) given in Definition 2.12. Since Θ is present inside floor functions in sbf and not so in lsbf, this substitution simplifies the computation. However, the resulting conditions are only sufficient, but not necessary, for schedulability.

Component	Workload	Utilization: $\left(\sum_i \frac{C_i}{T_i}\right)$
$\mathcal{C}_1$	$\{(45, 2, 45), (65, 3, 65), (85, 4, 85)\}$	0.137
$\mathcal{C}_2$	$\{(35, 2, 35), (55, 3, 55), (75, 4, 75)\}$	0.1653
$\mathcal{C}_3$	$\{(45, 1, 45), (75, 2, 75)\}$	0.049

Table 2.1: Example components scheduled on uniprocessor platform

[2] A smaller period value generally implies larger preemption overheads. However, a larger period value will result in higher bandwidth for the resource model. Designers can exploit this trade-off by specifying period values.



(a) Components $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$

(b) Components $\mathcal{C}_4$ and $\mathcal{C}_5$

Figure 2.8: Resource model plots: period vs. bandwidth

Example 2.1 *Consider the hierarchical system shown in Figure 2.4. Let the workload of components $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ be comprised of sporadic tasks as shown in Table 2.1. We obtained periodic resource model based abstractions ϕ_1, ϕ_2, and ϕ_3 for components $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, respectively, using techniques described in this section. The bandwidths of these resource models for varying period values are plotted in Figure 2.8(a). As can be seen from these plots and Table 2.1, bandwidth required to schedule these components using periodic models are significantly higher than the utilization of components. It can also be seen that the bandwidth increases with period value of resource models. We choose period values 5 and 7 for models ϕ_1 and ϕ_2 respectively. From Figure 2.4 we then get resource models $\phi_1 = \langle 5, 0.6912 \rangle$ and $\phi_2 = \langle 7, 1.6523 \rangle$. These are transformed to periodic tasks $\tau_{\phi_1} = (5, 0.6912, 5)$ and $\tau_{\phi_2} = (7, 1.6523, 7)$ that form the workload for component $\mathcal{C}_4$. Bandwidth of periodic resource model abstraction ϕ_4 is plotted in Figure 2.8(b). Similarly, choosing period values 10 and 6 for models ϕ_3 and ϕ_4 respectively, we get that the workload of component $\mathcal{C}_5$ is comprised of periodic tasks $\tau_{\phi_3} = (10, 0.5624, 10)$ and $\tau_{\phi_4} = (6, 4.1815, 6)$. The resulting periodic model bandwidth for component $\mathcal{C}_5$ is also plotted in Figure 2.8(b). Since there exists at least one period value for which bandwidth of ϕ_5 is at most one, the system shown in Figure 2.4 is schedulable on a uniprocessor platform using periodic resource models.*

Chapter 3

Related work

In this chapter we discuss past work related to the focus areas of this dissertation: schedulability analysis of hierarchical systems on uniprocessor platforms, and scheduling algorithms for multiprocessor platforms.

3.1 Hierarchical systems

Hierarchical systems were introduced by Deng and Liu [47], who proposed a two-level scheduling framework. Since then, many generalizations to this simple framework have been proposed, and schedulability analysis techniques for them have been developed. We now discuss these techniques and highlight their shortcomings.

On uniprocessor platforms, there has been a growing attention to compositional schedulability analysis of hierarchical systems [73, 79, 56, 95, 97, 80, 102, 101, 2, 42, 85, 113, 114, 107, 43, 3, 30, 58]. For the two-level system proposed by Deng and Liu, Kuo and Li [73] and Lipari *et. al.* [79, 81] presented exact schedulability conditions when higher level system scheduler is RM or EDF. Kuo and Li [73] considered RM system level scheduler, and assumed that tasks across all components have harmonic periods[1]. Lipari *et. al.* [79, 81] considered EDF system level scheduler,

[1]A set of numbers is harmonic if and only if each number in the set is either a divisor or a multiple of every

and assumed that the scheduler has knowledge of task deadlines in all components. Zhang and Burns [116] have recently extended these studies to consider arbitrary deadline task sets under EDF or DM system level scheduler. The common assumption shared by these approaches is that the system scheduler has exclusive access to the uniprocessor platform. For hierarchical systems, it is however desirable to be more general, because there could be more than two-levels of scheduling hierarchy and different schedulers may be used at different levels.

One such general approach was proposed by Regehr and Stankovic [95], where they considered various kinds of real-time guarantees. Their work focused on converting one kind of real-time guarantee into another. If a parent component's guarantee could be converted into the guarantee that one of its children demands, then they showed that the child is schedulable. However, they did not consider the problem of developing these real-time guarantees for components. In the following sections, we discuss two existing approaches that address this issue.

3.1.1 Resource model based compositional analysis

Resource models can be used to specify the real-time guarantees that a parent component provides to its children. Furthermore, since these models need not have exclusive access to the uniprocessor platform, they can also be used to analyze multi-level hierarchical systems. In the past, such schedulability analysis techniques have been proposed using two different resource models: bounded delay [90, 56, 101], and periodic [97, 80, 102, 2, 42].

Bounded delay resource model. Mok *et. al.* [90] introduced the bounded delay resource model that is specified using two parameters: a resource fraction c such that $0 \leq c \leq 1$, and a maximum delay δ. This model guarantees at least cL units of resource in every time interval of length $L+\delta$. It has been used for compositional schedulability analysis of hierarchical systems [56, 101], and Henzinger and Matic [67] have also extended these techniques to support incremental

other number in the set.

analysis.

Feng and Mok [56] extended the bounded-delay resource model with a third parameter: the minimum scheduling quantum Q. This model is similar to the original model, except that resource supply in the extended model can only be preempted at quantum boundaries. Shin and Lee [101] have used this extended model to abstract workloads of components. A major problem with using bounded delay models is in their transformation to tasks that can be used by higher level schedulers. Mok *et. al.* [90] have developed a transformation from bounded delay models to periodic tasks, but as illustrated by Shin and Lee [101], this transformation incurs resource overhead. To counter this overhead, Shin and Lee transformed each bounded delay model into an infinite sequence of jobs. Although this addresses the problem under EDF, it does not provide a solution under RM or DM schedulers.

Periodic resource model. Saewong *et. al.* [97], Lipari and Bini [80], and Shin and Lee [102], have introduced periodic resource models that characterize partial resource supplies with periodic allocation behaviors. To abstract the workload of a component using periodic models, Saewong *et. al.* [97] introduced an exact schedulability condition based on worst-case response time analysis, and Lipari and Bini [80] presented an exact condition based on time demand computations (rbf), both under fixed-priority schedulers. Likewise, under EDF, Shin and Lee [102] have presented an exact schedulability condition based on time demand computations (dbf). These studies have also presented trivial transformations from periodic resource models to tasks, thereby supporting compositional analysis of hierarchical systems.

All the aforementioned analysis techniques are not incremental, and hence do not support interface reuse. In Chapter 4 we address this problem and thus enable efficient on-line system adaption techniques and admission tests. Another problem with these previous approaches is that they incur resource overhead. To address this issue, in Chapter 5 we develop a new resource model that incurs smaller resource overhead than the periodic model. Also, in Chapter 6, we

quantify two notions of optimal resource utilization in hierarchical systems and develop workload abstractions that achieve one of them.

Dependency between components. Many studies have addressed the problem of handling inter- and intra-component dependencies in hierarchical systems [2, 42, 85, 43, 30, 58]. Almeida and Pedreiras [2] have presented compositional analysis techniques for the case when tasks in component workload have jitter in their releases. Davis and Burns [42] have extended this technique to consider release jitter as well as preemption overhead. Various resource-sharing protocols (HSRP [43], SIRAP [30], BROE [58]) that bound the maximum resource blocking time for dependent components have also been proposed in the past. However, all these approaches do not consider task offsets, which are often used to model communication dependencies between tasks (see [108]). Although these techniques can still be used for such task sets, the analysis will be pessimistic in general. In our case-study (see Chapter 7), we address this issue by developing exact schedulability conditions based on periodic resource models.

Matic and Henzinger [85] have also developed compositional analysis techniques in the presence of component dependencies. They assume dependencies are modeled using one of the following two semantics: *Real-time workshop (RTW)* [84], and *Logical execution time* [66]. Although RTW semantics is similar to the dependency constraints that we consider in our case study, it is more restrictive in that periods of dependent tasks are required to be harmonic.

3.1.2 Demand bound function based compositional analysis

Orthogonal to resource model based interface development, another body of work has focused on demand bound function (dbf) based workload abstractions [113, 114, 107, 3]. Recall that dbf gives the largest resource requirements of the workload for a given time interval length. Thiele *et. al.* [113, 114, 107] applied existing interface theories [44, 45] into real-time context by developing a real-time calculus [106] for these interfaces. The calculus defines a set of max-plus algebraic

operations on dbf, and this enables abstraction of component workloads under EDF and RM. In order to abstract workloads comprised of recurring branching tasks [18], similar dbf based compositional analysis techniques have been proposed by Anand *et. al.* [3].

In addition to being compositional, some of these techniques also support incremental analysis under EDF [114, 107]. However they are not incremental under RM, because they consider elements of the component workload in order of element priorities [113]. Another issue with this analysis is that popular schedulers like RM cannot schedule workloads abstracted as demand bound functions. They address this problem by transforming dbf's into periodic resource models, and as a result inherit all the shortcomings of periodic models that we described earlier.

3.2 Multiprocessor platforms

In general, studies on real-time multiprocessor scheduling theory can fall into two categories: *partitioned* and *global* scheduling. Under partitioned scheduling, each task is statically assigned to a single processor, and uniprocessor scheduling algorithms are used to schedule tasks. Under global scheduling, tasks are allowed to migrate across processors, and algorithms that simultaneously schedule on all the processors are used. Many partitioning algorithms and their analysis [93, 83, 26, 57], and global scheduling algorithms and their analysis [25, 6, 39, 105, 117, 62, 13, 19, 14, 16, 32, 38, 20, 40, 31, 27, 22, 21, 59], have been proposed in the past.

For implicit deadline task systems, both EDF [83] and RM [93] based partitioned scheduling have been proposed along-with resource utilization bounds. These studies have since been extended for constrained deadline task systems, and EDF [26] and fixed-priority [57] based scheduling have been developed for them. Under global scheduling of implicit deadline task systems, several optimal algorithms such as Pfair [25], BoundaryFair [117], LNREF [38], and NVNLF [59], have been proposed. To reduce the relatively high preemptions in these algorithms and to support

arbitrary deadline task systems, resource utilization bounds and worst-case response time analysis for EDF [62, 13, 14, 32, 20, 31, 22, 21] and DM [13, 16, 31, 27] based global scheduling strategies have been developed. Towards better resource utilization, new global scheduling algorithms such as dynamic-priority EDZL [39, 40] and US-EDF$\{m/(2m-1)\}$ [105], and fixed-priority RM-US$\{m/(3m-2)\}$ [6] and FP-EDF [19], have also been proposed. Partitioned scheduling suffers from an inherent performance limitation in that a task may fail to be assigned to any processor, although the total available processor capacity across the platform is larger than the task's requirements. Global scheduling has been developed to overcome this limitation. However, global algorithms are either not known to utilize resource optimally (like in the case of arbitrary deadline task systems), or if they are known to be optimal, then they have high preemption counts (like in the case of implicit deadline task systems). Moreover, for arbitrary deadline tasks, simulations conducted by Baker [15] have shown that partitioned scheduling performs much better than global scheduling on an average. These simulations reflect the large pessimism in current schedulability tests for global algorithms. To eliminate the performance limitation of partitioned scheduling, and to achieve high resource utilization without incurring high preemption costs, we consider the more general task-processor mappings that virtual cluster-based scheduling proposes (Chapter 8).

Algorithms that support slightly more general task-processor mappings than either partitioned or global scheduling have been proposed in the past. Andersson *et. al.* [9, 7, 8] and Kato and Yamasaki [71] have developed algorithms that allow a task to be scheduled on at most two processors in the platform. Baruah and Carpenter [23] introduced an approach that restricts processor migration of jobs, in order to alleviate the performance limitation of partitioned scheduling and the processor migration overhead of global scheduling. Their approach can be categorized as job-level partitioned scheduling. Virtual cluster-based scheduling framework that we propose in Chapter 8 generalizes all these task-processor mappings, and therefore can result in higher resource utilization. Calandrino *et. al.* [37] presented a physical clustering framework in which tasks are first

assigned to physical processor clusters, and then scheduled globally within those clusters. They experimentally evaluated this framework to show that cache-access related resource overhead can be reduced in comparison to both partitioned and global scheduling strategies. Virtual clustering is again a generalization of this framework, and moreover, unlike their work, we develop efficient schedulability analysis techniques with a focus on achieving high resource utilization. Recently, virtual clustering has also been considered in the context of tardiness guarantees for soft real-time systems [77].

Moir and Ramamurthy [87] and Anderson *et. al.* [68, 4] presented an approach that upper bounds the amount of concurrent execution within a group of tasks. They developed their approach using a two-level scheduling hierarchy under Pfair algorithm [25]. These studies are most related to our work on virtual clustering, but they differ from our technique mainly in the following aspect. We introduce a multiprocessor resource model that makes it possible to clearly separate intra-cluster and inter-cluster scheduling. This allows development of schedulability analysis techniques for virtual clustering that are easily extensible to many different schedulers. However, their approaches do not employ such a notion. Therefore their analysis techniques are bound to Pfair scheduling, and do not generalize to other algorithms and tasks such as the ones considered in Chapter 8. This flexibility provides a powerful tool for the development of various task-processor mappings and cluster scheduling algorithms.

Chapter 4

Incremental analysis of hierarchical systems

4.1 Introduction

Compositional schedulability analysis techniques based on resource models such as periodic ([80, 97, 102]) and bounded-delay (see [56, 101]) have been proposed in the past. One such technique based on the periodic model was described in Section 2.2.3. A major drawback of all these previous approaches is that component interfaces are not reusable. When the workload of a component changes, a new interface for that component cannot be generated using its existing interface and information on workload modifications.

Apart from increased flexibility, reusable interfaces also support efficient schedulability analysis of open systems (see [47, 114, 67, 45]). Open systems are systems in which components are partially specified, and their workloads can be modified on the fly. This gain in efficiency is particularly significant for hierarchical systems, because modifications could occur to components that are situated deep inside the hierarchy. Such analysis is useful for dynamic adaption of system parameters (*e.g.*, voltage scaling of processors [11, 96]), on-line admission tests of components

(see [114]), etc. However, the ability to reuse interfaces must go hand in hand with *incremental* analysis, so that the order in which elements of a workload are considered does not affect analysis. Otherwise, the same component could be declared either schedulable or not depending on the order of changes in the component's workload.

In this chapter we develop new component interfaces such that the resulting analysis technique is incremental. The interface model that we present is based on the periodic resource model [102, 80]. Each interface is comprised of a set of periodic resource models with period values in the range $1, \ldots, \Pi^*$, where Π^* is specified by system designer. In these interfaces we also account for preemption overhead incurred by components. Once interfaces are generated for all the components in a system, the designer can choose any resource model from a component's interface and use it to schedule the component's workload. In comparison to specifying period values a priori (see discussion in Section 2.2.3), this approach gives increased flexibility to the designer. He can choose resource models after taking into consideration multiple properties like resource bandwidth (resource used per unit time) and preemption overhead.

To generate interfaces for elementary components, we develop efficient algorithms under EDF and DM schedulers. Furthermore, to reduce storage space, we also develop compact representations for these interfaces. Examples show that the space required by these compact interfaces are small in practice. For non-elementary components, we develop compositional analysis techniques that use only associative functions on interface parameters. Since associative functions are independent of the order in which workload interfaces are considered, the resulting analysis is incremental. We first develop a technique that only considers those resource models from interfaces that have identical period values. Although this approach results in efficient analysis (minimum resource bandwidth), it also forces all components to be scheduled using resource models that have the same period. This restriction may not be acceptable under all circumstances. Therefore, we also develop another technique that considers any combination of resource models from interfaces.

Although the later technique eliminates single period restriction, it can only abstract workload interfaces scheduled under EDF. Finally, in comparison to the non-incremental technique described in Section 2.2.3, we derive an upper bound on the resource overhead incurred by our approach.

This chapter is organized as follows: In Section 4.2 we describe our assumptions for the system model and specify our problem statement. Section 4.3 introduces component interfaces used in this chapter, and Section 4.4 presents an interface generation algorithm for components that use EDF. Section 4.5 presents a similar algorithm for components that use DM. In Section 4.6 we present incremental analysis techniques using associative functions, and in Section 4.7 we derive an upper bound for the resource overhead incurred by our approach. We conclude the chapter and discuss future work in Section 4.8.

4.2 System assumptions and problem statement

In this chapter we assume that task sets in the workload of a component are specified using the periodic or sporadic task model described in Section 2.1.1. Although we focus on one task model, techniques presented here can be applied to any task model whose dbf[1] and rbf[2] are computable. For non-elementary components, we also assume that their workloads are comprised only of other components in the system, *i.e.*, they do not contain any tasks. This is a reasonable assumption, because many applications such as those found in avionics [55] can be modeled under this assumption. We also assume that the hierarchical systems we consider are scheduled on uniprocessor platforms.

As discussed in the introduction, workload of components in embedded systems are often modified on the fly. These modifications could happen either due to changes in user requirements, or due to changes in the operating environment. We assume that the only modifications possible

[1]Definition 2.6 in Section 2.1.5
[2]Definition 2.7 in Section 2.1.5

on a hierarchical system are: (1) addition of new components to the system, and (2) removal of existing components from the system. Note that workload changes for an existing component can be regarded as removal of the existing component and addition of a new component with the modified workload.

Preemptions play an important role in schedulability analysis because they consume resources. In a hierarchical system preemptions occur at each level of the hierarchy, *i.e.*, within each component. Therefore, in our analysis techniques it would be useful to account for preemption overhead incurred by components. We assume that preemption overheads incurred by elementary components are taken into account in their demand functions (dbf and rbf), using one of many existing techniques that upper bound this overhead (see [53, 35, 94]). Thus, we only consider preemption overheads incurred by non-elementary components. For the example shown in Figure 2.4, we assume that preemption overheads incurred by tasks in components $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ are taken into account in dbf and rbf of these components. We only consider overheads arising from preemptions that occur when components $\mathcal{C}_1$ and $\mathcal{C}_2$, or components $\mathcal{C}_3$ and $\mathcal{C}_4$, are scheduled under their parent's scheduler.

The schedulability analysis problem that we address in this chapter can be stated as follows: Given a hierarchical system,

1. Generate an interface for each elementary component.

2. Generate an interface for each non-elementary component such that the resulting analysis is incremental. Also, interfaces of workload components must take into account preemption overheads incurred by those components, when they are scheduled under the parent scheduler.

4.3 Component interface and its representation

4.3.1 Interface with multiple periodic resource models

In this work, each component interface is comprised of multiple periodic resource models, where each resource model has a different period value. As discussed in the introduction, such interfaces allow system designers to choose resource models considering a trade-off between different properties such as resource bandwidth and preemption overhead. Smaller period value generally reduces resource requirements for scheduling a component, but also increases preemption overhead. Since we are interested in minimizing overall resource usage of hierarchical systems, we carry these multiple resource models in interfaces. Resource usage can then be globally minimized once an interface is generated for the root component. We define our interface as follows.

Definition 4.1 (Multi-period interface) *A multi-period interface for a component $\mathcal{C}$ can be specified as $\mathcal{I}_\mathcal{C} = \{\phi_k = \langle k, \Theta_k \rangle | 1 \leq k \leq \Pi^*\}$, where Π^* is a system designer defined upper bound for period values. In this interface we restrict k to integer values.*

Section 2.2.2 gave schedulability conditions (Theorems 2.5 and 2.6) for elementary components scheduled using periodic resource models. These conditions can also be used to generate multi-period interfaces for elementary components. Let $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ denote a sporadic task set, and $\mathcal{C} = \langle \mathcal{T}, \mathcal{S} \rangle$ denote an elementary component. As described in Section 2.2.3, for each period value k, we can use Theorem 2.5 or 2.6 to compute the smallest capacity Θ_k required from model ϕ_k to schedule component $\mathcal{C}$. Note that sbf in these theorems is replaced with lsbf given in Definition 2.12 in Section 2.2.1. Since Theorem 2.5 can be evaluated in $\mathcal{O}(nL)$ time for each period value, the total time taken to generate interface $\mathcal{I}_\mathcal{C}$ under EDF is $\mathcal{O}(\Pi^* nL)$. Here, $L = \min\left\{\mathrm{LCM} + \max_{i=1,\ldots,n} \mathrm{D}_i, \frac{U_\mathcal{T}(\max_{i=1,\ldots,n}(\mathrm{T}_i - \mathrm{D}_i))}{1-U_\mathcal{T}}\right\}$, and LCM denotes the least common multiple of $\mathrm{T}_1, \ldots, \mathrm{T}_n$. Similarly, since Theorem 2.6 can be evaluated in $\mathcal{O}(n^2 \mathrm{D}_n)$ time for each period value, the total time taken to generate interface $\mathcal{I}_\mathcal{C}$ under DM is $\mathcal{O}(\Pi^* n^2 \mathrm{D}_n)$.

In Sections 4.4 and 4.5 we present more efficient algorithms to generate $\mathcal{I}_\mathcal{C}$ using these theorems. In particular, under EDF we generate the interface in $\mathcal{O}(n(\Pi^* + L))$ time, and under DM we generate it in $\mathcal{O}(n^2(\Pi^* + \mathrm{D}_n))$ time. Since we envision that our techniques will be used in on-the-fly analysis, these speed-ups can substantially improve the response times.

4.3.2 Compact representation for multi-period interfaces

Assuming each resource model can be represented using constant storage space, the amount of space required to store a multi-period interface is Π^*. In order to reduce this storage space, we use a compact representation for these interfaces. Consider interface $\mathcal{I}_\mathcal{C}$ described in the previous section. Since for each period k, Θ_k denotes the smallest resource capacity required by model ϕ_k to schedule component $\mathcal{C}$, ϕ_k exactly schedules $\mathcal{C}$ (see Theorems 2.5 and 2.6 in Section 2.2.2). If $\mathcal{C}$ uses EDF, this means that lsbf_{ϕ_k} intersects $\mathrm{dbf}_\mathcal{C}$ at some time instant t. If $\mathcal{C}$ uses DM, this means that for some $i \in \{1, \ldots, n\}$, $\mathrm{lsbf}_{\phi_k} \leq \mathrm{rbf}_{\mathcal{C},i}$ and lsbf_{ϕ_k} intersects $\mathrm{rbf}_{\mathcal{C},i}$ at some time instant t. ϕ_k can therefore be specified using its period k, this time instant t, and the value $\mathrm{dbf}_\mathcal{C}(t)$ or $\mathrm{rbf}_{\mathcal{C},i}(t)$. Theorem 2.5 or 2.6 can be used to generate Θ_k from this specification. This is the central idea for our compact representation of interfaces. This representation requires less space whenever lsbf's of different resource models intersect $\mathrm{dbf}_\mathcal{C}$ (or $\mathrm{rbf}_{\mathcal{C},i}$) at the same time instant. In Sections 4.4 and 4.5, we show using examples that storage requirements of such compact representations are much smaller than Π^*. We define this compact multi-period interface as follows.

Definition 4.2 (Compact multi-period interface) *A compact multi-period interface for component $\mathcal{C}$ can be specified as $\mathcal{CI}_\mathcal{C} = \{\langle j_{min}, j_{max}, t_j, d_j\rangle | j = 1, \ldots, l\}$, where l is a positive integer, $1_{min} = 1$ and $l_{max} = \Pi^*$. Furthermore, for each j, t_j and d_j are non-negative integers, $j_{min} \leq j_{max}$ and $(j+1)_{min} = j_{max} + 1$.*

$\mathcal{CI}_\mathcal{C}$ can be transformed into multi-period interface $\mathcal{I}_\mathcal{C}$ by computing resource capacity Θ_k for each resource model ϕ_k. For example, consider an element $\langle j_{min}, j_{max}, t_j, d_j\rangle$ of interface $\mathcal{CI}_\mathcal{C}$. In

this representation let t_j denote a time instant and d_j denote either $\text{dbf}_{\mathcal{C}}(t)$ or $\text{rbf}_{\mathcal{C},i}(t)$ for some $i \in 1, \ldots, n$. Periodic resource models $\phi_k = \langle k, \Theta_k \rangle$, with $k \in j_{min}, \ldots, j_{max}$, can then be obtained using either Equation (2.5) in Theorem 2.5 (if $\mathcal{C}$ uses EDF) or Equation (2.6) in Theorem 2.6 (if $\mathcal{C}$ uses DM). Since computation of each Θ_k can be done in constant time (lsbf is a quadratic function of Θ_k), the total time taken for this transformation is $\mathcal{O}(\Pi^*)$.

4.4 Interface generation under EDF scheduler

In this section we present an algorithm to generate compact multi-period interfaces for elementary components that use EDF. Consider a component $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$, where $\mathcal{T} = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_n = (\text{T}_n, \text{C}_n, \text{D}_n)\}$. For each period value k, $1 \leq k \leq \Pi^*$, we must compute the smallest capacity required from periodic resource model $\phi_k = \langle k, \Theta_k \rangle$ to schedule component $\mathcal{C}$. Let *relevant time instants* denote all the time instants at which some job of some task in $\mathcal{T}$ has a deadline. Note that $\text{dbf}_{\mathcal{C}}$ changes its value only at these time instants and it is a step function. Since ϕ_k exactly schedules $\mathcal{C}$, lsbf_{ϕ_k} must intersect $\text{dbf}_{\mathcal{C}}$ at one of these relevant time instants. Consider a naive algorithm which computes each model ϕ_k by walking through all the relevant time instants. This algorithm can generate interface $\mathcal{CI}_{\mathcal{C}}$ in $\mathcal{O}(\Pi^* nL)$ time, because each Θ_k can be computed in $\mathcal{O}(nL)$ time using Theorem 2.5 in Section 2.2.2 (after replacing sbf in the theorem with lsbf).

Algorithm 1 (EDF-COMPACT) computes $\mathcal{CI}_{\mathcal{C}}$ in $\mathcal{O}(n(\Pi^* + L))$ time using Theorem 2.5 (again sbf is replaced with lsbf), and is therefore faster than the naive algorithm by a factor of $\min\{\Pi^*, L\}$. EDF-COMPACT initially computes the minimum resource capacity Θ_1 required by resource model ϕ_1 (Line 1 of the algorithm). In Line 3 we check if the intersection time instant t_1 of lsbf_{ϕ_1} with $\text{dbf}_{\mathcal{C}}$ is equal to the earliest deadline in component $\mathcal{C}$. If yes, then $\mathcal{CI}_{\mathcal{C}}$ can be completely specified using t_1 and $\text{dbf}_{\mathcal{C}}(t_1)$. This is because $\text{dbf}_{\mathcal{C}}$ is zero for all time instants

Algorithm 1 EDF-COMPACT

Input: $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$, where $\mathcal{T} = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_n = (\text{T}_n, \text{C}_n, \text{D}_n)\}$.
Output: $\mathcal{CI}_{\mathcal{C}}$.

Compute $\Theta(= \Theta_1)$ using Theorem 2.5 with $\Pi = 1$ (replace sbf with lsbf).
Let $\text{lsbf}_{\phi_1}(t_1) = \text{dbf}_{\mathcal{C}}(t_1)$, where t_1 is the smallest such interval length and $\phi_1 = \langle 1, \Theta_1 \rangle$.
if $t_1 = \min_{i=1,\ldots,n}\{\text{D}_i\}$ **then**
    $\mathcal{CI}_{\mathcal{C}} \leftarrow \{\langle 1, \Pi^*, t_1, \text{dbf}_{\mathcal{C}}(t_1) \rangle\}$.
else
    Let $X = \langle x_1 = t_1, \ldots, x_N = \min_{i=1,\ldots,n}\{\text{D}_i\} \rangle$ such that for all i, $x_i > x_{i+1}$, and x_i is a relevant time instant.
    Let $Y = \langle y_1 = \text{dbf}_{\mathcal{C}}(x_1), \ldots, y_N = \text{dbf}_{\mathcal{C}}(x_N) \rangle$.
    Perform $\mathcal{L} = \langle l_1, \ldots, l_M \rangle \leftarrow$ SIG-INSTANTS(X, Y).
    Initialize $Z = 1, csi = x_{l_1}, nsi = x_{l_2}$, and $j = 2$.
    for $k = 2$ to Π^* **do**
        if $csi = \min_{i=1,\ldots,n}\{\text{D}_i\}$ **then**
            $\mathcal{CI}_{\mathcal{C}} \leftarrow \mathcal{CI}_{\mathcal{C}} \bigcup \{\langle Z, \Pi^*, csi, \text{dbf}_{\mathcal{C}}(csi) \rangle\}$.
        else
            Compute $\Theta(= \Theta'_k)$ using Equation (2.5) of Theorem 2.5 with $t = csi$ and $\Pi = k$ (replace sbf with lsbf).
            Compute $\Theta(= \Theta''_k)$ using Equation (2.5) of Theorem 2.5 with $t = nsi$ and $\Pi = k$ (replace sbf with lsbf).
            if $\Theta''_k > \Theta'_k$ **then**
                $\mathcal{CI}_{\mathcal{C}} \leftarrow \mathcal{CI}_{\mathcal{C}} \bigcup \{\langle Z, k-1, csi, \text{dbf}_{\mathcal{C}}(csi) \rangle\}$.
                Update $Z = k, j = j + 1, csi = nsi$ and $nsi = x_{l_j}$.
                Compute Θ using Equation (2.5) of Theorem 2.5 with $t = nsi$ and $\Pi = k$ (replace sbf with lsbf).
                while $\Theta = \Theta''_k$ **do**
                    Update $j = j + 1, csi = nsi$ and $nsi = x_{l_j}$.
                    Compute Θ using Equation (2.5) of Theorem 2.5 with $t = nsi$ and $\Pi = k$ (replace sbf with lsbf).
                end while
            end if
        end if
        if $k = \Pi^*$ **then**
            $\mathcal{CI}_{\mathcal{C}} \leftarrow \mathcal{CI}_{\mathcal{C}} \bigcup \{\langle Z, \Pi^*, csi, \text{dbf}_{\mathcal{C}}(csi) \rangle\}$.
        end if
    end for
end if

smaller than t_1, and then lsbf_{ϕ_k} for all $k > 1$ must intersect $\text{dbf}_{\mathcal{C}}$ at time instant t_1. For the other case, the algorithm computes a list $\mathcal{L}$ of indices using Procedure 2 (SIG-INSTANTS). List X passed to the procedure contains all relevant time instants smaller than or equal to t_1, such that they are sorted in decreasing order of time.

Procedure 2 SIG-INSTANTS

Input: $X = \langle x_1, \ldots, x_N \rangle, Y = \langle y_1 = \text{dbf}_{\mathcal{C}}(x_1), \ldots, y_N = \text{dbf}_{\mathcal{C}}(x_N) \rangle$.
Output: Index list $\mathcal{L}$.

1: Initialize $Z' = \{\langle 1, 2, 3 \rangle, \langle 2, 3, 4 \rangle, \ldots, \langle N-2, N-1, N \rangle\}$.
// Z' is the active index set. If $\langle i_1, i_2, i_3 \rangle \in Z'$, then $x_{i_2} \in X$ will be evaluated in current iteration.
2: Initialize $\mathcal{L} = \langle l_1 = 1, \ldots, l_N = N \rangle$.
3: **while** $Z' \neq \emptyset$ **do**
4: Initialize $Z = Z', Z' = \emptyset$.
5: **for** Each $\langle i_1, i_2, i_3 \rangle \in Z$ **do**
6: **if** (Slope of line connecting (x_{i_1}, y_{i_1}) and (x_{i_2}, y_{i_2})) $\geq$ (Slope of line connecting (x_{i_2}, y_{i_2}) and (x_{i_3}, y_{i_3})) **then**
7: Remove index i_2 from list $\mathcal{L}$.
// If $\text{lsbf}_{\phi_k}(x_{i_1}) = \text{dbf}_{\mathcal{C}}(x_{i_1})$ for some k, then no lsbf_{ϕ_j} for $j > k$ can intersect $\text{dbf}_{\mathcal{C}}$ at x_{i_2}. Hence, x_{i_2} cannot be a significant time instant.
// Now store tuples that need to be evaluated in next iteration.
8: **if** $\exists i$ such that i precedes i_1 in $\mathcal{L}$ **then**
9: $Z' \leftarrow Z' \bigcup \{\langle i, i_1, i_3 \rangle\}$.
10: **end if**
11: **if** $\exists i$ such that i succeeds i_3 in L **then**
12: $Z' \leftarrow Z' \bigcup \{\langle i_1, i_3, i \rangle\}$.
13: **end if**
14: **end if**
15: **end for**
16: **end while**
return $\mathcal{L}$.

SIG-INSTANTS takes as input the ordered list X and the corresponding values of $\text{dbf}_{\mathcal{C}}$, and outputs index list $\mathcal{L}$ that identifies certain special time instants in X. We call these special time instants *significant time instants*. A relevant time instant is also a significant time instant if and only if lsbf_{ϕ_k} for some $k \in \{1, \ldots, \Pi^*\}$ can intersect $\text{dbf}_{\mathcal{C}}$ at that time instant. We have identified a property that is useful in determining significant time instants. We explain this property with an example. Let x_{i_2} denote a relevant instant in the ordered list X. Then, x_{i_2} is also a significant instant if and only if there does not exist instants $x_{i_1}, x_{i_3} \in X$, such that $x_{i_1} < x_{i_2} < x_{i_3}$,

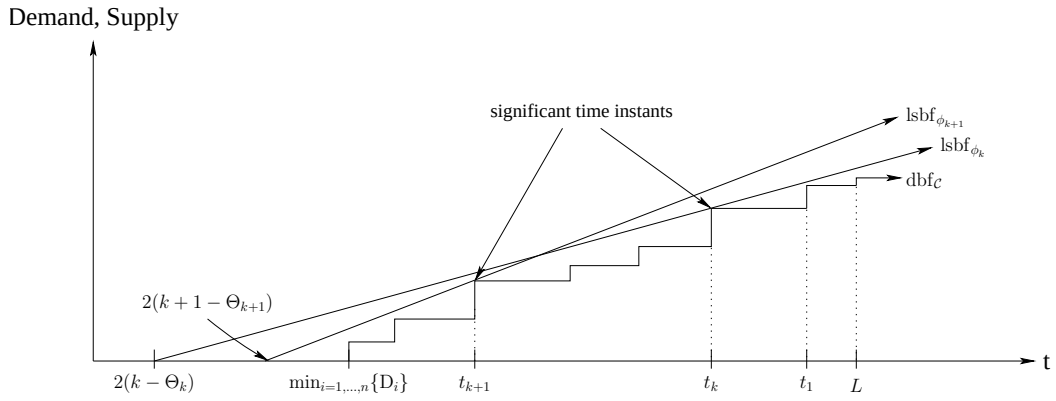


Figure 4.1: Properties of lsbf under EDF scheduler

and the line segment connecting points $(x_{i_1}, \text{dbf}_{\mathcal{C}}(x_{i_1}))$ and $(x_{i_3}, \text{dbf}_{\mathcal{C}}(x_{i_3}))$ lies above the point $(x_{i_2}, \text{dbf}_{\mathcal{C}}(x_{i_2}))$. Line 6 in SIG-INSTANTS checks this condition. In other words, points specified by significant time instants and their corresponding $\text{dbf}_{\mathcal{C}}$ values, form the *convex hull* of points specified by list X and their corresponding $\text{dbf}_{\mathcal{C}}$ values [3].

EDF-COMPACT then computes resource models ϕ_k for $k \in \{2, \ldots, \Pi^*\}$, using the significant time instants specified by $\mathcal{L}$. A nice property of significant time instants is that if lsbf_{ϕ_k} intersects $\text{dbf}_{\mathcal{C}}$ at some significant time instant say $x \in X$, then $\text{lsbf}_{\phi_{k+1}}$ is guaranteed to intersect $\text{dbf}_{\mathcal{C}}$ either at x or at the next significant time instant in X. Here we assume that x denotes the smallest time instant at which lsbf_{ϕ_k} intersects $\text{dbf}_{\mathcal{C}}$. This property is proved in Appendix A. For example, Figure 4.1 shows the lsbf's of two periodic resource models ϕ_k and ϕ_{k+1}. lsbf_{ϕ_k} intersects $\text{dbf}_{\mathcal{C}}$ at time instant t_k, and $\text{lsbf}_{\phi_{k+1}}$ intersects $\text{dbf}_{\mathcal{C}}$ at time instant t_{k+1}, where we assume that t_k and t_{k+1} are the smallest such time instants. As shown in the figure, t_k and t_{k+1} are both significant time instants. Also, $\text{lsbf}_{\phi_{k+1}}$ intersects $\text{dbf}_{\mathcal{C}}$ at all the significant time instants in interval (t_{k+1}, t_k). For each period value $k > 1$, EDF-COMPACT can therefore compute Θ_k by evaluating Equation (2.5) in Theorem 2.5 at exactly two significant time instants in X (Lines 14–15 in EDF-COMPACT). The while loop in Lines 20–23 ensures that we always

[3] Significant time instants can also be generated using Graham's scan algorithm [64] for convex hull computation. Since list X is already sorted and $\text{dbf}_{\mathcal{C}}$ is monotonic, we can show that this algorithm executes in linear time.

choose the earliest intersection point between lsbf_{ϕ_k} and $\text{dbf}_{\mathcal{C}}$.

Correctness of EDF-COMPACT is proved in Appendix A. We have shown that interface $\mathcal{CI}_{\mathcal{C}}$ generated by the algorithm represents minimum capacity resource models ϕ_k for all $k \in \{1, \ldots, \Pi^*\}$. We now prove that the running time of this algorithm is $\mathcal{O}(n(\Pi^* + L))$.

Theorem 4.1 EDF-COMPACT *runs in* $\mathcal{O}(n(\Pi^* + L))$ *time.*

Proof Line 1 of EDF-COMPACT executes in $\mathcal{O}(nL)$ time, where $L = \min\left\{\text{LCM} + \max_{i=1,\ldots,n} \text{D}_i, \frac{U_{\mathcal{T}}(\max_{i=1,\ldots,n}(\text{T}_i - \text{D}_i))}{1-U_{\mathcal{T}}}\right\}$ and LCM denotes the least common multiple of $\text{T}_1, \ldots, \text{T}_n$. This is the time it takes to evaluate Theorem 2.5 for all relevant time instants in the interval $(0, L]$, when sbf is replaced with lsbf. For each $k > 1$, EDF-COMPACT evaluates Equation (2.5) of Theorem 2.5 at exactly two time instants. Also, the while loop in Lines 20–23 will be executed at most $\mathcal{O}(L)$ times over all values of k. Hence, Lines 10–29 of the algorithm can be executed in $\mathcal{O}(n(\Pi^* + L))$ time. This follows from the fact that evaluating Equation (2.5) at one time instant takes $\mathcal{O}(n)$ time. Generating ordered list X in Line 6 can be done in $\mathcal{O}(nL)$ time by considering time instants in order from t_1 to $\min_{i=1,\ldots,n} \text{D}_i$. Let the call to procedure SIG-INSTANTS in Line 8 execute in $\mathcal{O}(L)$ time. These together imply that the running time of algorithm EDF-COMPACT is $\mathcal{O}(n(\Pi^* + L))$.

We now show that the call to procedure SIG-INSTANTS in executes in $\mathcal{O}(L)$ time. List X passed to the procedure has at most L time instants. Initially, set Z' has all the index tuples. Hence in the first iteration, $\mathcal{O}(L)$ slope comparisons are performed in Line 6 of SIG-INSTANTS. Set Z' is emptied at the beginning of each iteration. Furthermore, for every index that is removed from list $\mathcal{L}$ in an iteration, at most two tuples are added to set Z'. These tuples are used for slope comparisons in the next iteration. Therefore, the total number of slope comparisons performed in Line 6 after the first iteration, is bounded by a constant factor of the number of indices removed from $\mathcal{L}$. Since $\mathcal{L}$ is initialized with $\mathcal{O}(L)$ indices, the total number of slope comparisons performed

$\mathcal{CI}_{\mathcal{C}_1}$				$\mathcal{CI}_{\mathcal{C}_3}$			
j_{min}	j_{max}	t_j	d_j	j_{min}	j_{max}	t_j	d_j
1	1	9945	1369	1	6	225	11
2	4	2210	304	7	16	90	4
5	5	855	117	17	200	45	1
6	6	270	36				
7	21	90	11				
22	200	45	2				

Table 4.1: Compact multi-period interfaces $\mathcal{CI}_{\mathcal{C}_1}$ and $\mathcal{CI}_{\mathcal{C}_3}$

by SIG-INSTANTS is also bounded by $\mathcal{O}(L)$. Thus, we have shown that SIG-INSTANTS runs in $\mathcal{O}(L)$ time. □

Example 4.1 *Consider the hierarchical system shown in Figure 2.4. Let the workload of components* $\mathcal{C}_1$ *and* $\mathcal{C}_3$ *be comprised of tasks as shown in Table 2.1 in Section 2.2.3. We obtained multi-period interfaces for these components using* EDF-COMPACT*, where we assumed* $\Pi^* = 200$*. Compact interfaces* $\mathcal{CI}_{\mathcal{C}_1}$ *and* $\mathcal{CI}_{\mathcal{C}_3}$ *are given in Table 4.1, and interfaces* $\mathcal{I}_{\mathcal{C}_1}$ *and* $\mathcal{I}_{\mathcal{C}_3}$ *are plotted in Figure 2.8(a) in Section 2.2.3. It is clear from Table 4.1 that sizes of interfaces* $\mathcal{CI}_{\mathcal{C}_1}$ *and* $\mathcal{CI}_{\mathcal{C}_3}$ *are much smaller than* Π^**. We now describe how to obtain periodic resource models for interfaces* $\mathcal{I}_{\mathcal{C}_1}$ *and* $\mathcal{I}_{\mathcal{C}_3}$ *from their compact representations. Consider a period value say* $k = 10$*. Resource models for interfaces* $\mathcal{I}_{\mathcal{C}_1}$ *and* $\mathcal{I}_{\mathcal{C}_3}$ *can be computed using Table 4.1 and Equation* (2.5) *of Theorem 2.5 (*sbf *replaced with* lsbf*). For interface* $\mathcal{I}_{\mathcal{C}_1}$*, substituting* $\Pi = 10, t = 90$*, and* $\mathrm{dbf}_{\mathcal{C}_1}(t) = 11$ *in Equation* (2.5)*, we get* $\phi_{10} = \langle 10, 1.51 \rangle$*. Similarly, for interface* $\mathcal{I}_{\mathcal{C}_3}$*, putting* $\Pi = 10, t = 90$*, and* $\mathrm{dbf}_{\mathcal{C}_3}(t) = 4$ *in Equation* (2.5) *gives* $\phi_{10} = \langle 10, 0.562 \rangle$*.*

4.5 Interface generation under DM scheduler

In this section we describe algorithms to generate compact interfaces for elementary components that use DM. Consider a component $\mathcal{C} = \langle \mathcal{T}, \mathrm{DM} \rangle$, where $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n =$

$(\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$. We assume without loss of generality that τ_i has higher priority than τ_j for all $i < j$. We first describe an algorithm that generates a compact interface $\mathcal{CI}_{\mathcal{C},i}$ for each task τ_i in $\mathcal{C}$. We then show how to combine them in order to generate interface $\mathcal{CI}_{\mathcal{C}}$.

4.5.1 Interface for a single task in component workload

For each period value k, $1 \leq k \leq \Pi^*$,let $\phi_{k,i} = \langle k, \Theta_{k,i} \rangle$ denote the periodic resource model that exactly schedules task τ_i in component $\mathcal{C}$. To generate interface $\mathcal{CI}_{\mathcal{C},i}$ we must compute the resource capacity $\Theta_{k,i}$ for each k. Let *relevant time instants* denote a set of time instants at which some job of some task $\tau_j \in \mathcal{T}$, with $j \leq i$, is released. Note that $\mathrm{rbf}_{\mathcal{C},i}$ changes its value only at these time instants and it is a step function. Then, similar to the EDF case, $\mathrm{lsbf}_{\phi_{k,i}}$ must intersect $\mathrm{rbf}_{\mathcal{C},i}$ at one of these relevant time instants. For each k, resource capacity $\Theta_{k,i}$ can then be computed using Theorem 2.6 from Section 2.2.2 in $\mathcal{O}(i\,\mathrm{D}_i)$ time. Hence, interface $\mathcal{CI}_{\mathcal{C},i}$ can be computed in $\mathcal{O}(\Pi^*\,i\,\mathrm{D}_i)$ time using a naive algorithm that repeatedly considers all relevant time instants for each k.

Algorithm 3 (DM-TASK) uses an approach based on *significant time instants*, which is similar to the approach used by EDF-COMPACT. DM-TASK computes interface $\mathcal{CI}_{\mathcal{C},i}$ in $\mathcal{O}(i(\Pi^* + \mathrm{D}_i))$ time, and hence is more efficient than the aforementioned naive algorithm. It first computes model $\phi_{1,i}$ using Theorem 2.6 (Line 1 of the algorithm). For period values greater than 1, the algorithm computes an index list $\mathcal{L}$ using procedure SIG-INSTANTS given in Section 4.4. The ordered list X passed to this procedure contains all relevant time instants greater than or equal to t_1 (Line 8 of DM-TASK), where t_1 is the largest time instant with property $\mathrm{lsbf}_{\phi_{1,i}}(t_1) = \mathrm{rbf}_{\mathcal{C},i}(t_1)$. Under DM, time instant $x_{j_2} \in X$ is a significant time instant if and only if there does not exist a pair of time instants $x_{j_1}, x_{j_3} \in X$, such that $x_{j_1} < x_{j_2} < x_{j_3}$, and the line segment connecting points $(x_{j_1}, \mathrm{rbf}_{\mathcal{C},i}(x_{j_1}))$ and $(x_{j_3}, \mathrm{rbf}_{\mathcal{C},i}(x_{j_3}))$ lies below the point $(x_{j_2}, \mathrm{rbf}_{\mathcal{C},i}(x_{j_2}))$. Again, points specified by significant time instants and their corresponding $\mathrm{rbf}_{\mathcal{C},i}$

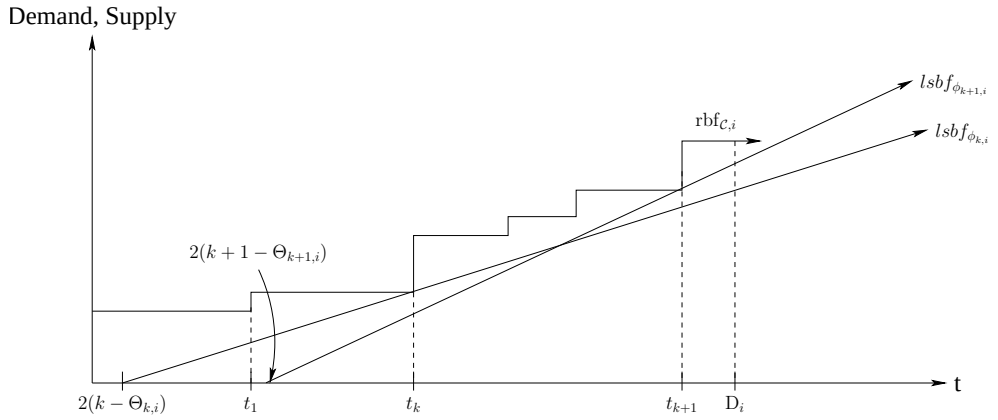


Figure 4.2: Properties of lsbf under DM scheduler

values, form the *convex hull* of points specified by list X and their corresponding $\text{rbf}_{\mathcal{C},i}$ values.

DM-TASK then computes resource models $\phi_{k,i}$ for all $k > 1$, using the significant time instants in X. For each k, the algorithm is required to evaluate Equation (2.6) of Theorem 2.6 at only two significant time instants. This can be explained using an example. Figure 4.2 shows lsbf's of two resource models $\phi_{k,i}$ and $\phi_{k+1,i}$. If $\text{lsbf}_{\phi_{k,i}}$ intersects $\text{rbf}_{\mathcal{C},i}$ at time instant t_k and $\text{lsbf}_{\phi_{k+1,i}}$ intersects $\text{rbf}_{\mathcal{C},i}$ at time instant t_{k+1}, then t_k and t_{k+1} are both significant time instants (assuming these are the largest such time instants). Furthermore, $\text{lsbf}_{\phi_{k+1,i}}$ intersects $\text{rbf}_{\mathcal{C},i}$ at all the significant time instants in the interval (t_k, t_{k+1}).

Line 1 of DM-TASK executes in $\mathcal{O}(i\,\text{D}_i)$ time. Since the list X passed in Line 8 to procedure SIG-INSTANTS has $\mathcal{O}(\text{D}_i)$ time instants, from Theorem 4.1 we get that this call executes in $\mathcal{O}(\text{D}_i)$ time. Also, the while loop in Lines 20–23 execute at most $\mathcal{O}(\text{D}_i)$ times over all values of k. Thus, DM-TASK computes interface $\mathcal{CI}_{\mathcal{C},i}$ in $\mathcal{O}(i(\Pi^* + \text{D}_i))$ time. Proof of correctness of algorithm DM-TASK is given in Appendix B.

Algorithm 3 DM-TASK

Input: $\mathcal{C} = \langle \mathcal{T}, \text{DM} \rangle$, where $\mathcal{T} = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_n = (\text{T}_n, \text{C}_n, \text{D}_n)\}$.
Output: Interface $\mathcal{CI}_{\mathcal{C},i}$ for task τ_i.

1: Compute $\Theta(= \Theta_{1,i})$ using Theorem 2.6 with $\Pi = 1$ (replace sbf with lsbf).
2: Let $\text{lsbf}_{\phi_{1,i}}(t_1) = \text{rbf}_{\mathcal{C},i}(t_1)$, where t_1 is the largest such interval length and $\phi_{1,i} = \langle 1, \Theta_{1,i} \rangle$.
3: **if** $t_1 = \text{D}_i$ **then**
4: $\quad \mathcal{CI}_{\mathcal{C},i} \leftarrow \{\langle 1, \Pi^*, t_1, \text{rbf}_{\mathcal{C},i}(t_1) \rangle\}$.
5: **else**
6: $\quad$ Let $X = \langle x_1 = t_1, \ldots, x_N = \text{D}_i \rangle$ such that for all i, $x_i < x_{i+1}$, and x_i is a relevant time instant.
7: $\quad$ Let $Y = \langle y_1 = \text{rbf}_{\mathcal{C},i}(x_1), \ldots, y_N = \text{rbf}_{\mathcal{C},i}(x_N) \rangle$.
8: $\quad$ Perform $\mathcal{L} = \langle l_1, \ldots, l_M \rangle \leftarrow$ SIG-INSTANTS(X, Y).
9: $\quad$ Initialize $Z = 1, csi = x_{l_1}, nsi = x_{l_2}, j = 2$.
10: $\quad$ **for** $k = 2$ to Π^* **do**
11: $\quad\quad$ **if** $csi = \text{D}_i$ **then**
12: $\quad\quad\quad \mathcal{CI}_{\mathcal{C},i} \leftarrow \mathcal{CI}_{\mathcal{C},i} \bigcup \{\langle Z, \Pi^*, csi, \text{rbf}_{\mathcal{C},i}(csi) \rangle\}$.
13: $\quad\quad$ **else**
14: $\quad\quad\quad$ Compute $\Theta(= \Theta'_{k,i})$ using Equation (2.6) of Theorem 2.6 with $t = csi$ and $\Pi = k$ (replace sbf with lsbf).
15: $\quad\quad\quad$ Compute $\Theta(= \Theta''_{k,i})$ using Equation (2.6) of Theorem 2.6 with $t = nsi$ and $\Pi = k$ (replace sbf with lsbf).
16: $\quad\quad\quad$ **if** $\Theta''_{k,i} > \Theta'_{k,i}$ **then**
// In this case, lsbf ***of resource models with period*** $> k$ ***will intersect*** $\text{rbf}_{\mathcal{C},i}$ ***at time instants*** $\geq nsi$***. Hence, update*** $\mathcal{CI}_{\mathcal{C},i}$***.***
17: $\quad\quad\quad\quad \mathcal{CI}_{\mathcal{C},i} \leftarrow \mathcal{CI}_{\mathcal{C},i} \bigcup \{\langle Z, k-1, csi, \text{rbf}_{\mathcal{C},i}(csi) \rangle\}$.
18: $\quad\quad\quad\quad$ Update $Z = k, j = j + 1, csi = nsi$ and $nsi = x_{l_j}$.
19: $\quad\quad\quad\quad$ Compute Θ using Equation (2.6) of Theorem 2.6 with $t = nsi$ and $\Pi = k$ (replace sbf with lsbf).
20: $\quad\quad\quad\quad$ **while** $\Theta = \Theta''_{k,i}$ **do**
21: $\quad\quad\quad\quad\quad$ Update $j = j + 1, csi = nsi$ and $nsi = x_{l_j}$.
22: $\quad\quad\quad\quad\quad$ Compute Θ using Equation (2.6) of Theorem 2.6 with $t = nsi$ and $\Pi = k$ (replace sbf with lsbf).
23: $\quad\quad\quad\quad$ **end while**
24: $\quad\quad\quad$ **end if**
25: $\quad\quad$ **end if**
26: $\quad\quad$ **if** $k = \Pi^*$ **then**
27: $\quad\quad\quad \mathcal{CI}_{\mathcal{C},i} \leftarrow \mathcal{CI}_{\mathcal{C},i} \bigcup \{\langle Z, \Pi^*, csi, \text{rbf}_{\mathcal{C},i}(csi) \rangle\}$.
28: $\quad\quad$ **end if**
29: $\quad$ **end for**
30: **end if**

4.5.2 Interface for entire component workload

We assume that multi-period interfaces $\mathcal{CI}_{\mathcal{C},1}, \ldots, \mathcal{CI}_{\mathcal{C},n}$ for tasks $\tau_1, \ldots, \tau_n$ respectively, have been generated using algorithm DM-TASK. In this section we combine them to generate interfaces $\mathcal{I}_\mathcal{C}$ and $\mathcal{CI}_\mathcal{C}$ for component $\mathcal{C}$.

Let $\mathcal{I}_{\mathcal{C},1}, \ldots, \mathcal{I}_{\mathcal{C},n}$ denote multi-period interfaces for tasks $\tau_1, \ldots, \tau_n$ respectively, such that these interfaces are obtained from their compact representations. Let $GiveModel_k(\mathcal{I}_{\mathcal{C},i})$ return resource model $\phi_{k,i} = \langle k, \Theta_{k,i} \rangle$ and $GiveBandwidth_k(\mathcal{I}_{\mathcal{C},i})$ return resource bandwidth $\frac{\Theta_{k,i}}{k}$. Algorithm 4 (DM-COMPACT) then computes interface $\mathcal{I}_\mathcal{C}$. For each period k, DM-COMPACT first identifies the task τ_l that requires maximum resource capacity, *i.e.*, it identifies resource model $\phi_{k,l}$ that has largest bandwidth among models $\phi_{k,1}, \ldots, \phi_{k,n}$. $\phi_{k,l} = \langle k, \Theta_{k,l} \rangle$ then becomes the resource model in interface $\mathcal{I}_\mathcal{C}$. This follows from the fact that resource models must satisfy Theorem 2.6 from Section 2.2.2 in order to schedule component $\mathcal{C}$. By definition, $\Theta_{k,l}$ is the smallest capacity that satisfies this equation. Compact interface $\mathcal{CI}_\mathcal{C}$ can be obtained from interface $\mathcal{I}_\mathcal{C}$ using compact interfaces $\mathcal{CI}_{\mathcal{C},1}, \ldots, \mathcal{CI}_{\mathcal{C},n}$. For a period k, let τ_l denote the task that required maximum resource bandwidth. The tuple in interface $\mathcal{CI}_\mathcal{C}$ that represents resource model for period k, can then be trivially obtained from the corresponding tuple in interface $\mathcal{CI}_{\mathcal{C},l}$; t_j and d_j values are identical in the two tuples.

Transforming each $\mathcal{CI}_{\mathcal{C},i}$ to interface $\mathcal{I}_{\mathcal{C},i}$ can be done in $\mathcal{O}(\Pi^*)$ time. Algorithm DM-COMPACT runs in $\mathcal{O}(n\,\Pi^*)$ time, and interface $\mathcal{CI}_\mathcal{C}$ can be obtained from interface $\mathcal{I}_\mathcal{C}$ in $\mathcal{O}(n\,\Pi^*)$ time. Also, computation of each interface $\mathcal{CI}_{\mathcal{C},i}$ takes $\mathcal{O}(i(\Pi^* + \mathrm{D}_i))$ time. Therefore, the total time taken to generate interface $\mathcal{CI}_\mathcal{C}$ is $\mathcal{O}(n^2(\Pi^* + \mathrm{D}_n))$.

Example 4.2 *Consider the hierarchical system shown in Figure 2.4. Let the workload of component $\mathcal{C}_2$ be comprised of tasks as shown in Table 2.1 in Section 2.2.3. We obtained multi-period*

Algorithm 4 DM-COMPACT

Input: $\mathcal{C} = \langle \mathcal{T}, \text{DM} \rangle$, where $\mathcal{T} = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_n = (\text{T}_n, \text{C}_n, \text{D}_n)\}$.
Input: Interfaces $\mathcal{I}_{\mathcal{C},1}, \ldots, \mathcal{I}_{\mathcal{C},n}$.
Output: $\mathcal{I}_{\mathcal{C}}$.

1: **for** $k = 1$ to Π^* **do**
   // Identify resource model $\phi_{k,l}$ that requires largest capacity among resource models $\phi_{k,1}, \ldots, \phi_{k,n}$.
2: Compute $l = \arg\max_{i=1,\ldots,n} GiveBandwidth_k(\mathcal{I}_{\mathcal{C},i})$.
3: $\mathcal{I}_{\mathcal{C}} \leftarrow \mathcal{I}_{\mathcal{C}} \bigcup \{GiveModel_k(\mathcal{I}_{\mathcal{C},l})\}$.
4: **end for**

$\mathcal{CI}_{\mathcal{C}_2}$			
j_{min}	j_{max}	t_j	d_j
1	22	70	14
23	200	35	2

Table 4.2: Compact multi-period interface $\mathcal{CI}_{\mathcal{C}_2}$

interfaces for this component using algorithms DM-TASK *and* DM-COMPACT*, where we assumed* $\Pi^* = 200$*. Interface* $\mathcal{CI}_{\mathcal{C}_2}$ *is given in Table 4.2, and interface* $\mathcal{I}_{\mathcal{C}_2}$ *is plotted in Figure 2.8(a) in Section 2.2.3. It is easy to see that the size of interface* $\mathcal{CI}_{\mathcal{C}_2}$ *is much smaller than* Π^**.*

4.6 Incremental analysis

Algorithms presented in previous sections generate multi-period interfaces for elementary components. Resource models in these interfaces must be transformed into tasks so that higher level parent schedulers can schedule them. Additionally, we must also develop techniques to compose these interfaces with each other, in order to generate interfaces for non-elementary components. One way to achieve this composition is to transform non-elementary components into elementary components using the resource model to task transformation, so that algorithms presented in previous sections can again be used. However, this approach does not support incremental analysis, because schedulability conditions presented in Theorems 2.5 and 2.6 do not use associative functions on task parameters. Even otherwise, these theorems cannot reuse existing interfaces of

components to generate new interfaces that abstract modified workloads. Therefore in this section, we first present techniques that directly compose resource models in multi-period interfaces using only associative functions. We then transform the resource models into tasks such that resource requirements under this transformation are consistent with the proposed composition techniques.

4.6.1 Interface composition using associative functions

Let $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \mathcal{S} \rangle$ denote a non-elementary component. For all i, $1 \leq i \leq n$, we assume that multi-period interface $\mathcal{I}_{\mathcal{C}_i}$ has been generated using algorithms presented in previous sections. If some component $\mathcal{C}_i$ itself is non-elementary, then we assume that its interface has been generated using techniques developed here. In each interface $\mathcal{I}_{\mathcal{C}_i}$, we denote the resource model corresponding to period value k as $\phi_{k,i} = \langle k, \Theta_{k,i} \rangle$. Also, let $\phi_k = \langle k, \Theta_k \rangle$ denote the resource model in interface $\mathcal{I}_{\mathcal{C}}$. We now develop a technique to compute capacity Θ_k using associative functions on models $\phi_{k,1}, \ldots, \phi_{k,n}$. This composition can be defined as follows.

Definition 4.3 (Multi-period composition (identical periods)) *Let* $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \mathcal{S} \rangle$ *denote a non-elementary component and for all* i, $1 \leq i \leq n$, *let* $\mathcal{I}_{\mathcal{C}_i} = \bigcup_{k=1}^{\Pi^*} \{\phi_{k,i} = \langle k, \Theta_{k,i} \rangle\}$. *Then, interface* $\mathcal{I}_{\mathcal{C}} = \bigcup_{k=1}^{\Pi^*} \{\phi_k = \langle k, \Theta_k \rangle\}$ *is such that*

$$\forall k, 1 \leq k \leq \Pi^*, \Theta_k = \sum_{i=1}^{n} \Theta_{k,i}$$

Since addition is associative, this composition supports incremental analysis, *i.e.*, order of composition of interfaces $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$ does not affect interface $\mathcal{I}_{\mathcal{C}}$. Compact interface $\mathcal{CI}_{\mathcal{C}}$ can be generated from compact representations $\mathcal{CI}_{\mathcal{C}_1}, \ldots, \mathcal{CI}_{\mathcal{C}_n}$ as follows. For each i, let t_i and d_i denote elements in a tuple of compact representation $\mathcal{CI}_{\mathcal{C}_i}$, such that this tuple corresponds to period Π. Also, let t and d denote elements in a tuple of compact representation $\mathcal{CI}_{\mathcal{C}}$, such that

this tuple also corresponds to period Π. Then, $\text{lsbf}_{\phi_{k,i}}(t_i) = d_i$ for each i, and we can set d to $\sum_{i=1}^{n} d_i$ and compute t using equation $\sum_{i=1}^{n} \text{lsbf}_{\phi_{k,i}}(t_i) = \text{lsbf}_{\phi_k}$.

We now describe how this composition enables multi-period interfaces to be reused. In the first example a new component is added to the workload of $\mathcal{C}$, and in the second example an existing component in the workload of $\mathcal{C}$ is modified.

1. Let the workload of $\mathcal{C}$ change such that a new component $\mathcal{C}_{n+1}$ is added to its workload. Then, the new interface for $\mathcal{C}$ can be directly obtained from its existing interface and interface $\mathcal{I}_{\mathcal{C}_{n+1}} = \bigcup_{k=1}^{\Pi^*} \{\phi_{k,n+1} = \langle k, \Theta_{k,n+1} \rangle\}$. For each period k set $\Theta_k = \Theta_k + \Theta_{k,n+1}$.

2. Let the workload of some component $\mathcal{C}_i$ change so that interface $\mathcal{I}_{\mathcal{C}_i}$ gets modified to interface $\mathcal{I}'_{\mathcal{C}_i} = \bigcup_{k=1}^{\Pi^*} \{\phi'_{k,i} = \langle k, \Theta'_{k,i} \rangle\}$. Then, the new interface for $\mathcal{C}$ can be directly obtained from its existing interface, and interfaces $\mathcal{I}_{\mathcal{C}_i}$ and $\mathcal{I}'_{\mathcal{C}_i}$. For each period k set $\Theta_k = \Theta_k - \Theta_{k,i} + \Theta'_{k,i}$.

Note that interface reuse substantially improves efficiency of on-the-fly schedulability analysis. New interface for component $\mathcal{C}$ can be generated in $\mathcal{O}(\Pi^*)$ time as described above. Both under EDF and DM this improvement is significant, because the complexity of algorithm EDF-COMPACT is $\mathcal{O}(n(\Pi^* + L))$ and that of algorithms DM-TASK and DM-COMPACT is $\mathcal{O}(n^2(\Pi^* + \text{D}_n))$. We now prove that this composition is correct in that it preserves schedulability. We show that the minimum amount of resource supply provided by resource model ϕ_k, is at least as much as the minimum amount of resource supply collectively required for models $\phi_{k,1}, \ldots, \phi_{k,n}$.

Theorem 4.2 *Let interfaces $\mathcal{I}_{\mathcal{C}}$ and $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$ be as specified in Definition 4.3. Then for all k, $1 \leq k \leq \Pi^*$, $\text{sbf}_{\phi_k} \geq \sum_{i=1}^{n} \text{sbf}_{\phi_{k,i}}$.*

Proof For all i, $1 \leq i \leq n$, let $s_i = k - 2\,\Theta_k + \sum_{j=1}^{i} \Theta_{k,j}$. Consider the function

$$\text{sbf}_i = \begin{cases} \left\lfloor \frac{t-s_i}{k} \right\rfloor \Theta_{k,i} + \max\left\{0, t - 2(k - \Theta_k) - \sum_{j=1}^{i-1} \Theta_{k,j} - \left\lfloor \frac{t-s_i}{k} \right\rfloor k\right\} & t \geq s_i \\ 0 & \text{Otherwise} \end{cases}$$

Also, recall from Equation (2.3) in Section 2.2.1 that

$$\mathrm{sbf}_{\phi_{k,i}}(t) = \begin{cases} \left\lfloor \frac{t-(k-\Theta_{k,i})}{k} \right\rfloor \Theta_{k,i} + \max\left\{0, t - 2(k-\Theta_{k,i}) - \left\lfloor \frac{t-(k-\Theta_{k,i})}{k} \right\rfloor k\right\} & t \geq k - \Theta_{k,i} \\ 0 & \text{Otherwise} \end{cases}$$

Observe that $s_i \leq k - \Theta_{k,i}$. If $\left\lfloor \frac{t-s_i}{k} \right\rfloor > \left\lfloor \frac{t-(k-\Theta_{k,i})}{k} \right\rfloor$, then since the term inside max function in $\mathrm{sbf}_{\phi_{k,i}}$ is at most $\Theta_{k,i}$, we get $\mathrm{sbf}_i \geq \mathrm{sbf}_{\phi_{k,i}}$. If $\left\lfloor \frac{t-s_i}{k} \right\rfloor = \left\lfloor \frac{t-(k-\Theta_{k,i})}{k} \right\rfloor$, then since $2\,\Theta_{k,i} \leq 2\,\Theta_k - \sum_{j=1}^{i-1} \Theta_{k,j}$, in this case as well we get $\mathrm{sbf}_i \geq \mathrm{sbf}_{\phi_{k,i}}$.

Thus, we have shown that $\mathrm{sbf}_i \geq \mathrm{sbf}_{\phi_{k,i}}$ for all i. But by definition, $\mathrm{sbf}_{\phi_k} = \sum_{i=1}^{n} \mathrm{sbf}_i$. Therefore, $\mathrm{sbf}_{\phi_k} \geq \sum_{i=1}^{n} \mathrm{sbf}_{\phi_{k,i}}$. □

We can repeat this composition at each level of a hierarchical system until a multi-period interface is generated for the root component. To schedule individual components in the system, the designer can choose some period value say k^*. This choice can be made taking into consideration some desired property such as minimization of resource bandwidth. Each component in the system can then be scheduled using that resource model in its interface whose period is k^*. Theorems 2.5, 2.6, and 4.2 guarantee that the component is schedulable using this resource model.

Although this composition is incremental, there are several issues with it. A major problem is that we have not specified any transformation from resource models to tasks. In order that Theorem 4.2 remains valid, this transformation must also ensure that no overhead is incurred, *i.e.*, amount of resource required to schedule the task must be equal to sbf of corresponding resource model. In addition to this issue, there are two problems with the composition technique: (1) it does not take into account preemption overhead incurred by components, and (2) its correctness relies on the assumption that all components in the system are scheduled using resource models with identical periods. In the following section we account for preemption overhead in component interfaces. In Section 4.6.3 we develop a new composition technique under EDF scheduler that does not rely on assumption (2). In that section we also develop a transformation from resource

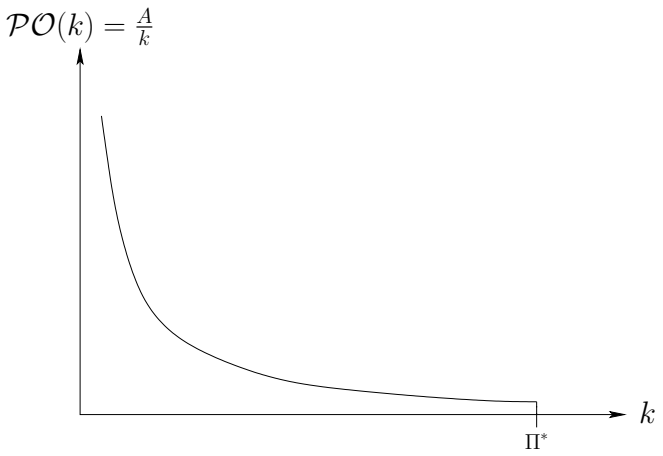


Figure 4.3: Preemption overhead function

models to tasks that is consistent with Theorem 4.2.

4.6.2 Interface modification for preemption overhead

Consider the non-elementary component $\mathcal{C}$ described in previous section. We now modify each resource model $\phi_{k,i}$ to account for preemption overhead incurred by $\mathcal{C}_i$, when it executes under $\mathcal{C}$'s scheduler.

Many techniques have been proposed to upper bound the number of preemptions incurred by a set of periodic tasks when they are scheduled under EDF or DM (see [35, 94, 53]). These can be used to bound the preemption overhead of resource models in interface $\mathcal{I}_{\mathcal{C}_i}$, after the resource models are transformed to tasks. However, in open environments, it is often impossible for a component to know beforehand the other components that will be scheduled with it. This means that the resource models in $\mathcal{I}_{\mathcal{C}_i}$ need not be aware of resource models in interfaces $\mathcal{I}_{\mathcal{C}_j}$, for $j \neq i$. Therefore, we cannot use these previously developed techniques to bound preemption overhead.

Lipari and Bini have presented a technique to estimate preemption overhead in periodic resource models [80], and we will use this technique in our framework. They use any convex and decreasing function over periods to estimate preemption overhead. This choice of function is motivated by the assumption that as period value increases, the resource models are preempted less

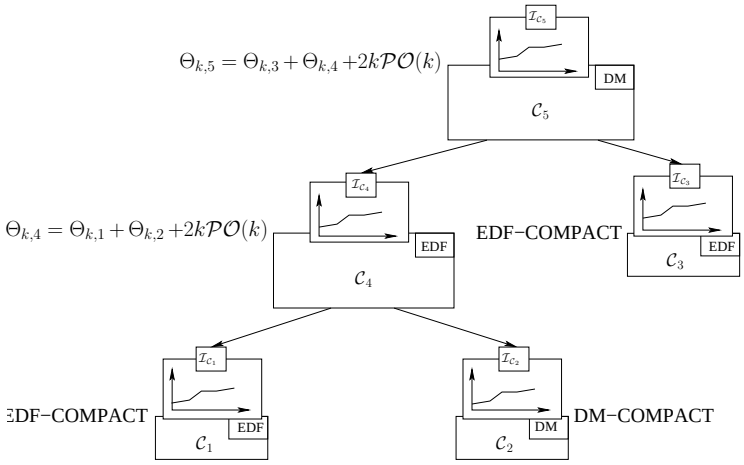


Figure 4.4: Illustration of analysis techniques

number of times. Let $\mathcal{PO} : \mathbb{Z}^+ \longrightarrow \Re$ denote our convex preemption estimate function. Given a period value, $\mathcal{PO}$ gives the preemption overhead for any component scheduled using a periodic resource model having that period. Furthermore, without loss of generality, we also assume that this function gives the overhead per unit time. For example, Figure 4.3 shows such a function ($\mathcal{PO}(k) = \frac{A}{k}$), where A is a constant specified by the system designer and k denotes resource model period. It is easy to see that $\mathcal{PO}$ given in the figure is a convex function, *i.e.*, for all i and j, $\mathcal{PO}(i) + \mathcal{PO}(j) \geq \mathcal{PO}(i + j)$. To account for preemption overhead in interface $\mathcal{I}_{\mathcal{C}_i}$, we transform each model $\phi_{k,i} = \langle k, \Theta_{k,i} \rangle$ to the model $\phi'_{k,i} = \langle k, \Theta_{k,i} + k\mathcal{PO}(k) \rangle$.

Thus to compose interfaces $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$, we first transform all the resource models in these interfaces to account for preemption overhead. The transformed interfaces are then composed using Definition 4.3 given in the previous section. This process of resource model transformation and interface composition can be repeated at each level of the hierarchical system, until a multi-period interface is generated for the root component.

Example 4.3 *Consider the hierarchical system shown in Figure 2.4. An illustration of our interface generation and composition techniques for this system is shown in Figure 4.4. As shown in*

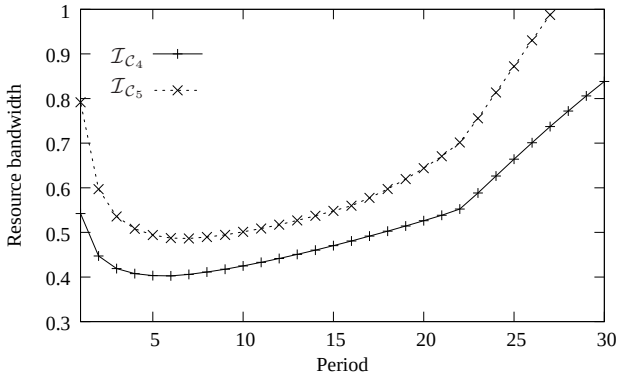


Figure 4.5: Multi-period interfaces for components $\mathcal{C}_4$ and $\mathcal{C}_5$

the figure, interfaces $\mathcal{I}_{\mathcal{C}_1}$,$\mathcal{I}_{\mathcal{C}_2}$, and $\mathcal{I}_{\mathcal{C}_3}$ are generated using algorithms EDF-COMPACT, DM-TASK, *and* DM-COMPACT. *These interfaces are also discussed in Examples 4.1 and 4.2. To generate an interface for component $\mathcal{C}_4$, we compose interfaces $\mathcal{I}_{\mathcal{C}_1}$ and $\mathcal{I}_{\mathcal{C}_2}$. For this purpose we assume that $A = 0.1$ in function $\mathcal{PO}$. Resource models in interfaces $\mathcal{I}_{\mathcal{C}_1}$ and $\mathcal{I}_{\mathcal{C}_2}$ are first transformed using function $\mathcal{PO}$, and then composed using Definition 4.3 to generate interface $\mathcal{I}_{\mathcal{C}_4}$. Resource bandwidth of models in $\mathcal{I}_{\mathcal{C}_4}$ are plotted in Figure 4.5 for period values in the range* 1 *to* 30. *Similarly, to generate interface $\mathcal{I}_{\mathcal{C}_5}$ we compose interfaces $\mathcal{I}_{\mathcal{C}_3}$ and $\mathcal{I}_{\mathcal{C}_4}$. The bandwidth of resource models in $\mathcal{I}_{\mathcal{C}_5}$ are also plotted in Figure 4.5. As shown in the figure, resource model with period* 7 *($\langle 7, 3.4048\rangle$) has the smallest bandwidth in interface $\mathcal{I}_{\mathcal{C}_5}$. Since this smallest bandwidth is* $0.4864(\leq 1)$, *the hierarchical system of Figure 2.4 is schedulable on an uniprocessor platform using our technique. If the designer chooses* 7 *as desired period value, then components in this system can be scheduled using those resource models in their interfaces that have period* 7.

4.6.3 Improved composition under EDF scheduler

A major drawback of the composition technique described in Section 4.6.1 is that it only composes resource models with identical period values. This restriction may not be always acceptable. For instance, period values of resource models may be specified a priori by system designers (*e.g.*, avionics system described in Chapter 7). Therefore, in this section, we present a technique that

can compose periodic resource models having different periods. This technique also supports incremental analysis, but can only be used to compose interfaces scheduled under EDF.

Consider a non-elementary component $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \text{EDF} \rangle$. To compose a resource model $\phi = \langle \Pi, \Theta \rangle$ from interface $\mathcal{I}_{\mathcal{C}_i}$ of component $\mathcal{C}_i$, we use two parameters associated with it: (1) resource bandwidth $\frac{\Theta}{\Pi}$, and (2) point at which lsbf_ϕ intersects time axis, *i.e.*, blackout interval $2(\Pi - \Theta)$. This composition is defined as follows.

Definition 4.4 (Multi-period composition (different periods)) *Let* $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \text{EDF} \rangle$ *denote a non-elementary component and for all* i, $1 \leq i \leq n$, *let* $\mathcal{I}_{\mathcal{C}_i}$ *denote the interface for component* $\mathcal{C}_i$. *Let* $\mathcal{I}_{\mathcal{C}} = \bigcup_{k=1}^{\Pi^*} \{\phi_k = \langle k, \Theta_k \rangle\}$ *denote the interface for component* $\mathcal{C}$. *Also, for each* k, $1 \leq k \leq \Pi^*$, *let* $k_1, \ldots, k_n$ *denote the designer specified periods of resource models in interfaces* $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$ *that must be composed. If* $\phi_{k_i} = \langle k_i, \Theta_{k_i} \rangle$ *denotes the resource model in interface* $\mathcal{I}_{\mathcal{C}_i}$ *for each* i, *then* Θ_k *is the smallest value satisfying following inequalities.*

$$\frac{\Theta_k}{k} \geq \sum_{i=1}^{n} \frac{\Theta_{k_i}}{k_i} + \mathcal{PO}(k_i)$$

$$2(k - \Theta_k) \leq \frac{\min_{i=1,\ldots,n}\{2(k_i - \Theta_{k_i} - k_i \mathcal{PO}(k_i))\}}{2}$$

In the above definition there is no restriction on the different k_i's, and hence we can compose resource models with different periods. Also, note that the composition accounts for preemption overhead incurred by components. As before, we now describe how this composition enables interface reuse using two examples.

1. Let the workload of component $\mathcal{C}$ change such that a new component $\mathcal{C}_{n+1}$ is added to its workload. Then, the new interface for $\mathcal{C}$ can be directly obtained from its existing interface and interface $\mathcal{I}_{\mathcal{C}_{n+1}}$. For each period value k, let k' denote the period of resource model in

interface $\mathcal{I}_{\mathcal{C}_{n+1}}$ that must be composed, and let $\phi_{k'} = \langle k', \Theta_{k'} \rangle \in \mathcal{I}_{\mathcal{C}_{n+1}}$. Then, choose a new resource capacity Θ'_k for each resource model ϕ_k in $\mathcal{I}_{\mathcal{C}}$ such that $\frac{\Theta'_k}{k} \geq \frac{\Theta_k}{k} + \frac{\Theta_{k'}}{k'} + \mathcal{PO}(k')$ and $2(k - \Theta'_k) \leq \min\{2(k - \Theta_k), k' - \Theta_{k'} - k'\mathcal{PO}(k')\}$.

2. Let the workload of some component $\mathcal{C}_i$ change so that interface $\mathcal{I}_{\mathcal{C}_i}$ gets modified to interface $\mathcal{I}'_{\mathcal{C}_i}$. Also, for each k, let $\phi'_{k_i} = \langle k_i, \Theta'_{k_i} \rangle$ denote the resource model in interface $\mathcal{I}'_{\mathcal{C}_i}$ that must be composed. Then, the new interface for $\mathcal{C}$ can be directly obtained from its existing interface, interfaces $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$, and interface $\mathcal{I}'_{\mathcal{C}_i}$. Choose a new resource capacity Θ'_k for each resource model ϕ_k in $\mathcal{I}_{\mathcal{C}}$ such that $\frac{\Theta'_k}{k} \geq \frac{\Theta_k}{k} - \frac{\Theta_{k_i}}{k_i} + \frac{\Theta'_{k_i}}{k_i}$ and $2(k - \Theta'_k) \leq \min\{\min_{j \neq i}\{k_j - \Theta_{k_j} - k_j\mathcal{PO}(k_j)\}, k_i - \Theta'_{k_i} - k_i\mathcal{PO}(k_i)\}$. Although this composition is required to use interfaces of all components in the workload of $\mathcal{C}$, it is nevertheless more efficient than using algorithm EDF-COMPACT.

We now prove this composition is correct in that it preserves schedulability of interfaces.

Theorem 4.3 *Let interfaces $\mathcal{I}_{\mathcal{C}}$ and $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$ be as specified in Definition 4.4. Also, for each resource model ϕ_{k_i} in $\mathcal{I}_{\mathcal{C}_i}$, let $\phi'_{k_i} = \langle k_i, \Theta_{k_i} + k_i\mathcal{PO}(k_i) \rangle$. Then for each k, $1 \leq k \leq \Pi^*$, there exists periodic tasks $\tau_{k_1}, \ldots, \tau_{k_n}$ such that*

1. *for each i, $1 \leq i \leq n$, the amount of resource required to satisfy the demand of task τ_{k_i} is at least as much as $\mathrm{sbf}_{\phi'_{k_i}}$, and*

2. *component $\mathcal{C} = \langle \{\tau_{k_1}, \ldots, \tau_{k_n}\}, \mathrm{EDF} \rangle$ is schedulable using periodic resource model ϕ_k.*

Proof For each ϕ'_{k_i}, consider the periodic task $\tau_{k_i} = (k_i, \Theta_{k_i} + k_i\mathcal{PO}(k_i), 2k_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i))$. We now show that the amount of resource required to satisfy the demand of τ_{k_i} is at least as much as $\mathrm{sbf}_{\phi'_{k_i}}$. In particular, we show that any resource model R that can satisfy the resource requirements of τ_{k_i} also satisfies the condition $\mathrm{sbf}_R \geq \mathrm{sbf}_{\phi'_{k_i}}$. For illustration purposes $\mathrm{dbf}_{\tau_{k_i}}$ and $\mathrm{sbf}_{\phi'_{k_i}}$ are plotted in Figure 4.6. We consider two different types of interval lengths: one where

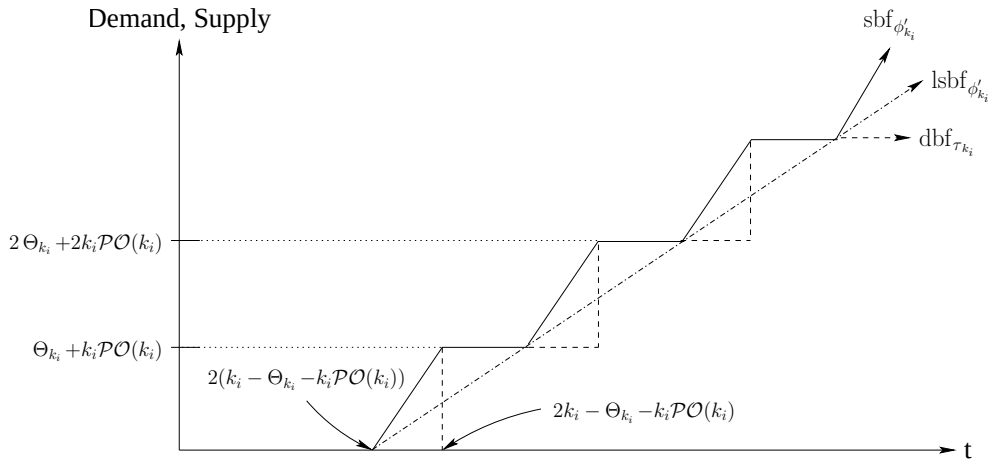


Figure 4.6: Demand of task τ_{k_i} and $\mathrm{sbf}_{\phi'_{k_i}}$

the length is in the range $(nk_i - 2\,\Theta_{k_i} - 2k_i\mathcal{PO}(k_i), nk_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i))$ and the other where it is in the range $[nk_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i), (n+1)k_i - 2\,\Theta_{k_i} - 2k_i\mathcal{PO}(k_i)]$, for all $n \geq 2$. Informally, these intervals correspond to the rising and flat portions of $\mathrm{sbf}_{\phi'_{k_i}}$ respectively.

Case 1: $t \in (nk_i - 2\,\Theta_{k_i} - 2k_i\mathcal{PO}(k_i), nk_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i))$

We prove this case by contradiction. Let $\mathrm{sbf}_R(t) < \mathrm{sbf}_{\phi'_{k_i}}(t)$. For interval length $t' = nk_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i)$ we have $\mathrm{sbf}_R(t') \geq \mathrm{dbf}_{\tau_{k_i}}(t')$, because resource model R satisfies the demand of task τ_{k_i}. Also, $\mathrm{sbf}_{\phi'_{k_i}}(t') = \mathrm{dbf}_{\tau_{k_i}}(t')$ by definition. These together imply $\mathrm{sbf}_{\phi'_{k_i}}(t') - \mathrm{sbf}_{\phi'_{k_i}}(t) < \mathrm{sbf}_R(t') - \mathrm{sbf}_R(t)$. This is a contradiction, because in interval $[t, t']$, $\mathrm{sbf}_{\phi'_{k_i}}$ is always rising with unit slope and no sbf for uniprocessor platforms can rise with a greater slope.

Case 2: $t \in [nk_i - \Theta_{k_i}, (n+1)k_i - 2\,\Theta_{k_i} - 2k_i\mathcal{PO}(k_i)]$

In this case $\mathrm{dbf}_{\tau_{k_i}}(t) = \mathrm{sbf}_{\phi'_{k_i}}(t)$. Also, any resource model R that supplies enough resource to satisfy the demand of τ_{k_i} must satisfy the condition $\mathrm{sbf}_R(t) \geq \mathrm{dbf}_{\tau_{k_i}}(t)$. Hence, $\mathrm{sbf}_R(t) \geq \mathrm{sbf}_{\phi'_{k_i}}(t)$.

Combining Cases 1 and 2 above we get $\mathrm{sbf}_R(t) \geq \mathrm{sbf}_{\phi'_{k_i}}(t)$ for all $t \geq 2k_i - 2\,\Theta_{k_i} - 2k_i\mathcal{PO}(k_i)$. Since $\mathrm{sbf}_{\phi'_{k_i}}(t) = 0$ for all $t < 2k_i - 2\,\Theta_{k_i} - 2k_i\mathcal{PO}(k_i)$, we have proved that the amount of resource required to schedule τ_{k_i} is at least as much as $\mathrm{sbf}_{\phi'_{k_i}}$. Going a step further, since $\mathrm{dbf}_{\tau_{k_i}} \leq \mathrm{sbf}_{\phi'_{k_i}}$,

it is easy to see that any resource R with $\text{sbf}_R \geq \text{sbf}_{\phi'_{k_i}}$ can satisfy the demand of task τ_{k_i}, *i.e.*, this transformation is exact.

We now show that component $\mathcal{C}$ is schedulable using periodic resource model ϕ_k. For each i, $1 \leq i \leq n$, consider the function

$$\text{usbf}_i(t) = \left(\frac{\Theta_{k_i}}{k_i} + \mathcal{PO}(k_i)\right)(t - (k_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i)))$$

Clearly $\text{usbf}_i \geq \text{sbf}_{\phi'_{k_i}} \geq \text{dbf}_{\tau_{k_i}}$. Then

$$\begin{aligned}
\sum_{i=1}^{n} \text{dbf}_{\tau_{k_i}} &\leq \sum_{i=1}^{n} \text{usbf}_i \\
&= \sum_{i=1}^{n} \left[\left(\frac{\Theta_{k_i}}{k_i} + \mathcal{PO}(k_i)\right)(t - (k_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i)))\right] \\
&< \left(t - \min_{i=1}^{n}\{k_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i)\}\right) \sum_{i=1}^{n} \left[\frac{\Theta_{k_i}}{k_i} + \mathcal{PO}(k_i)\right] \\
&= \text{lsbf}_{\phi_k} && \text{By definition} \\
&\leq \text{sbf}_{\phi_k}
\end{aligned}$$

From Theorem 2.5 in Section 2.2.2 we then get that component $\mathcal{C}$ is schedulable using resource model ϕ_k. This proves the theorem. □

To generate interface $\mathcal{I}_\mathcal{C}$ we can compose interfaces $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$ using Definition 4.4, and this composition can be done in $\mathcal{O}(\Pi^*)$ time.

Discussion. Although the worst-case resource demand of each periodic task τ_{k_i} is sufficient to satisfy $\text{sbf}_{\phi'_{k_i}}$, task τ_{k_i} itself cannot be directly used for distributing resource from the parent component to interface ϕ'_{k_i}. Instead, the following dynamic job dispatch scheme can be used. A job of task τ_{k_i} is dispatched for execution at the same time instant as the relative finish time of previous job, if the finish time is smaller than k_i. Otherwise, the job is dispatched at its regular, relative dispatch time k_i. Since the parent component's interface supplies resource satisfying $\text{dbf}_{\tau_{k_i}}$, it always guarantees at least $\Theta_{k_i} + k_i\mathcal{PO}(k_i)$ resource units in any time interval of length

$2k_i - \Theta_{k_i} - k_i\mathcal{PO}(k_i)$). Therefore, the parent interface can schedule the dynamic job dispatches described above. A point to note is that this mapping from resource models to periodic tasks can also be used in the composition described in Section 4.6.1. This follows from the fact that this mapping is both necessary and sufficient.

Example 4.4 *Again consider the hierarchical system shown in Figure 2.4. We now analyze component* $\mathcal{C}_4$ *using the composition technique developed in this section. For this purpose we assume that interfaces* $\mathcal{I}_{\mathcal{C}_1}$ *and* $\mathcal{I}_{\mathcal{C}_2}$ *are generated using algorithms* EDF-COMPACT, DM-TASK, *and* DM-COMPACT. *These interfaces are discussed in Examples 4.1 and 4.2. We assume that* $A = 0.1$ *in function* $\mathcal{PO}$*, and then compose these interfaces using Definition 4.4. We consider all possible combinations of resource models in* $\mathcal{I}_{\mathcal{C}_1}$ *and* $\mathcal{I}_{\mathcal{C}_2}$ *for this composition. This means, for each period* k*, we consider all possible values of* k_1 *and* k_2 *and choose the smallest resource capacity among them for model* ϕ_k *in* $\mathcal{I}_{\mathcal{C}_4}$*. Bandwidth of these resource models in* $\mathcal{I}_{\mathcal{C}_4}$ *are plotted in Figure 4.7 for period values in the range* 1 *to* 30*.*

Comparing these bandwidths to the ones plotted in Figure 4.5, we can see that the composition presented here is not only more general, but also more efficient for this particular example. Also, from Figure 4.7 we can see that the bandwidth increases sharply beyond period 13*. This is because condition (2) in Definition 4.4 becomes more significant than condition (1) for all period values greater than* 13*. The smallest bandwidth among resource models in interface* $\mathcal{I}_{\mathcal{C}_4}$ *is obtained when period is* 11*. This model is specified as* $\langle 11, 4.3988\rangle$*. The corresponding resource models in interfaces* $\mathcal{I}_{\mathcal{C}_1}$ *and* $\mathcal{I}_{\mathcal{C}_2}$ *are* $\langle 17, 3.1147\rangle$ *and* $\langle 20, 6.6089\rangle$ *respectively. From Theorems 4.3 and 2.5 we get that components* $\mathcal{C}_1, \mathcal{C}_2$*, and* $\mathcal{C}_4$ *are schedulable using these resource models.*

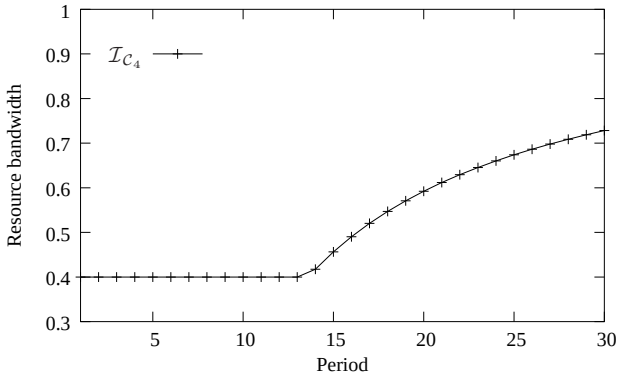


Figure 4.7: Multi-period interface for component $\mathcal{C}_4$

4.7 Comparison

In this section we compare the incremental approach described in Sections 4.4, 4.5, and 4.6.1 (henceforth denoted as INC), with the non-incremental approach presented by Shin and Lee [102] and described in Section 2.2.3 (henceforth denoted as NINC). For this purpose, we consider a hierarchical system comprised of n elementary and k non-elementary components, and compare a resource bandwidth upper bound of INC with a bandwidth lower bound of NINC. Let

- $\mathcal{C}_1, \ldots, \mathcal{C}_n$ denote the n elementary components,
- $\text{LOAD}_{\mathcal{C}_1}, \ldots, \text{LOAD}_{\mathcal{C}_n}$ denote the schedulability loads of $\mathcal{C}_1, \ldots, \mathcal{C}_n$, respectively,
- $\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}$ denote multi-period interfaces for components $\mathcal{C}_1, \ldots, \mathcal{C}_n$, respectively, generated using algorithms EDF-COMPACT, DM-TASK, and DM-COMPACT,
- $B_{i,j}$ denote the bandwidth of resource model in interface $\mathcal{I}_{\mathcal{C}_i}$ for period value j,
- $\mathcal{PO}(\Pi) = \frac{A}{\Pi}$ denote the preemption overhead function, where A is a constant and Π is the resource model period,
- $B_i (= \min_j \{B_{i,j} + \mathcal{PO}(j)\})$ denote the minimum bandwidth over all resource models in interface $\mathcal{I}_{\mathcal{C}_i}$, taking into account preemption overhead, and

- $\Pi_i (= \arg\min_j \{B_{i,j} + \mathcal{PO}(j)\})$ denote the resource model period corresponding to the minimum bandwidth. Without loss of generality, we assume $\Pi_k \leq \Pi_l$ for all $k < l$.

- $\mathcal{C}_{n+1}, \ldots, \mathcal{C}_{n+k}$ denote the k non-elementary components,

NINC assumes that resource model periods for interfaces are specified a priori by system designer. Since we are interested in a lower bound for the bandwidth overhead of NINC, we assume it chooses periods that locally minimize bandwidths. In other words, we assume NINC (1) chooses resource models with periods $\Pi_1, \ldots, \Pi_n$ as interfaces for components $\mathcal{C}_1, \ldots, \mathcal{C}_n$ respectively, (2) transforms each resource model $\phi = \langle \Pi, \Theta \rangle$ into the periodic task $\tau_\phi = (\Pi, \Theta, \Pi)$, (3) generates multi-period interfaces for each non-elementary component using EDF-COMPACT, or DM-TASK and DM-COMPACT, and (4) chooses the resource model with minimum bandwidth as interface for each non-elementary component, after considering preemption overhead as in Section 4.6.2.

4.7.1 Resource bandwidth upper bound for INC

INC is sub-optimal with respect to resource usage for two reasons; identical period restriction on resource models (interface composition in Section 4.6.1), and resource overhead incurred by elementary component interfaces generated using algorithms EDF-COMPACT and DM-COMPACT (*abstraction overhead*). For any specific period value, these algorithms generate resource models that are identical to the ones generated by NINC. Hence, bandwidth upper bounds derived in this earlier work can be used to bound the abstraction overhead incurred by INC.

Multi-period interfaces composed using Theorem 4.2 in Section 4.6.1 are all required to use resource models with identical period values. As a result, interface of elementary component $\mathcal{I}_{\mathcal{C}_i}$ may be forced to use a resource model whose period is not Π_i, *i.e.*, a resource model whose

bandwidth is not the minimum. We now derive an upper bound on the resource overhead incurred due to this sub-optimal choice. For this purpose, we derive an upper bound on the bandwidth of resource model with period Π_1, that is generated by INC for the root component. Since INC eventually selects a resource model with smallest bandwidth for the root component, the aforementioned bound is also an upper bound for this selection.

Theorem 4.4 *Let B_{inc} denote the minimum bandwidth among all resource models in the root component interface generated by* INC*. Let B' denote the bandwidth of resource model with period Π_1 in this interface. Then, $B_{inc} \leq B' \leq \sum_{i=1}^{n} B_{i,\Pi_i} + \frac{A(n+k)}{\Pi_1}$.*

Proof For period Π_1, preemption overhead for all the k non-elementary components is $k\mathcal{PO}(\Pi_1) = k\frac{A}{\Pi_1}$. Then, from Definition 4.3 we get

$$B' = B_{1,\Pi_1} + \frac{A}{\Pi_1} + \sum_{i=2}^{n} \left(B_{i,\Pi_1} + \frac{A}{\Pi_1} \right) + k\frac{A}{\Pi_1} \tag{4.1}$$

We now upper bound B_{i,Π_1} for all $i \neq 1$. Since $\Pi_1 \leq \Pi_i$, using Lemma A.1 in Appendix A (or) Lemma B.1 in Appendix B we get $B_{i,\Pi_1} \leq B_{i,\Pi_i}$. Using this bound in Equation (4.1) we get,

$$\begin{aligned} B' \leq & B_{1,\Pi_1} + \frac{A}{\Pi_1} + \frac{Ak}{\Pi_1} + \sum_{i=2}^{n} (B_{i,\Pi_i} + \frac{A}{\Pi_1}) \\ = & \sum_{i=1}^{n} B_{i,\Pi_i} + \frac{A(n+k)}{\Pi_1} \end{aligned}$$

Since the minimum bandwidth generated by INC must be smaller than the bandwidth for some particular period value, we get $B_{inc} \leq B' \leq \sum_{i=1}^{n} B_{i,\Pi_i} + \frac{A(n+k)}{\Pi_1}$. □

4.7.2 Bandwidth lower bound for EDF-COMPACT

Although Shin and Lee [102] have derived an upper bound for the bandwidth overhead of algorithms EDF-COMPACT and DM-COMPACT, we also require a lower bound for these overheads in order to compare INC with NINC. In this section, we first derive these lower bounds for an elementary component $\mathcal{C}$, assuming it is scheduled using periodic resource model $\phi = \langle \Pi, \Theta \rangle$.

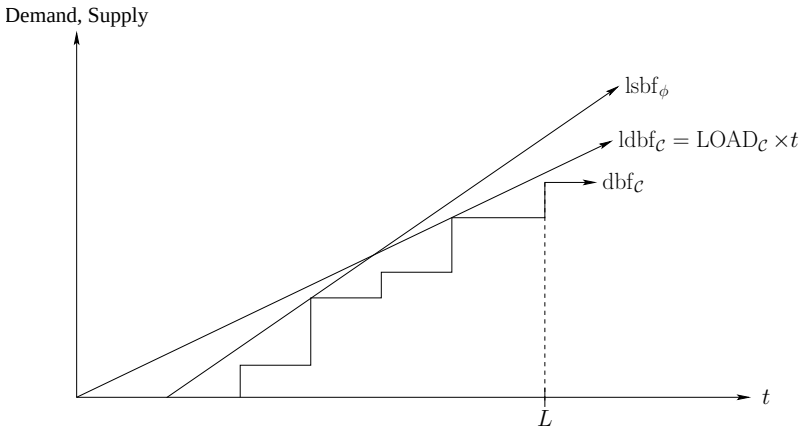


Figure 4.8: Bandwidth lower bound

We then show that a resource model with period one incurs the smallest overhead and derive the corresponding bound.

EDF is an optimal scheduling algorithm for sporadic task systems that are being considered in this chapter [29]. Suppose lsbf_ϕ intersects $\text{dbf}_\mathcal{C}$ for interval length t. Then, since EDF is optimal, $\text{lsbf}_\phi(t) >= \text{dbf}_\mathcal{C}(t)$ is also a necessary condition for resource model ϕ to schedule component $\mathcal{C}$ under any other scheduling algorithm. In other words, the bandwidth required by ϕ to schedule $\mathcal{C}$ under any scheduling algorithm, is at least as much as the bandwidth required by ϕ to schedule $\mathcal{C}$ under EDF. Since we are computing a lower bound on the bandwidth overhead, we assume that $\mathcal{C}$ uses EDF.

Consider function $\text{ldbf}_\mathcal{C} = t \times \text{LOAD}_\mathcal{C}$ shown in Figure 4.8. This function is a linear upper bound of the demand bound function $\text{dbf}_\mathcal{C}$. Also, there exists t such that $0 < t \leq L$ and $\text{ldbf}_\mathcal{C}(t) = \text{dbf}_\mathcal{C}(t)$, where $L = \min\left\{\text{LCM} + \max_{i=1}^{n} \text{D}_i, \frac{U_\mathcal{T}(\max_{i=1}^{n}(\text{T}_i - \text{D}_i))}{1-U_\mathcal{T}}\right\}$ and LCM denotes the least common multiple of periods of tasks in $\mathcal{C}$'s workload. The following theorem gives a lower bound for the bandwidth overhead incurred by algorithms EDF-COMPACT and DM-COMPACT. It uses the fact that for $\mathcal{C}$ to be schedulable using resource model ϕ, $\text{lsbf}_\phi(L) \geq \text{ldbf}_\mathcal{C}(L)$. This follows from the fact that $\text{lsbf}_\phi(t) \geq \text{ldbf}_\mathcal{C}(t) = \text{dbf}_\mathcal{C}(t)$.

Theorem 4.5 *Given period* Π *and component* $\mathcal{C}$ *that uses* EDF *scheduler, bandwidth overhead*

$BO_{\mathcal{C}}(\Pi)$ incurred by resource models generated using algorithms EDF-COMPACT *and* DM-COMPACT *is bounded by,*

$$BO_{\mathcal{C}}(\Pi) \geq \frac{-A + \sqrt{A^2 + 16\,\Pi^2\,\mathrm{LOAD}_{\mathcal{C}}(1 - \mathrm{LOAD}_{\mathcal{C}})}}{4\,\Pi},$$

where $A = (L - 2\,\Pi + 4\,\Pi \times \mathrm{LOAD}_{\mathcal{C}})$. Also, the minimum value for this overhead is attained when $\Pi = 1$ and

$$BO_{\mathcal{C}}(1) \geq \frac{\mathrm{LOAD}_{\mathcal{C}}(1 - \mathrm{LOAD}_{\mathcal{C}})}{L + 2},$$

assuming $L \geq \max\{2 + 4 \times \mathrm{LOAD}_{\mathcal{C}}(1 - 2 \times \mathrm{LOAD}_{\mathcal{C}}), 2 + \sqrt{2} - 4 \times \mathrm{LOAD}_{\mathcal{C}}\}$.

Proof Theorem 2.5 in Section 2.2.2 gives the schedulability condition for component $\mathcal{C}$ using resource model ϕ. Since $\mathrm{ldbf}_{\mathcal{C}}(L) \geq \mathrm{dbf}_{\mathcal{C}}(L)$ and $\mathrm{ldbf}_{\mathcal{C}}(t) = \mathrm{dbf}_{\mathcal{C}}(t)$ for some t such that $0 < t \leq L$, for $\mathcal{C}$ to be schedulable using ϕ,

$$\mathrm{lsbf}_{\phi}(L) \geq \mathrm{ldbf}_{\mathcal{C}}(L) = L \times \mathrm{LOAD}_{\mathcal{C}}$$

$$\Rightarrow \frac{\Theta}{\Pi}(L - 2(\Pi - \Theta)) \geq L \times \mathrm{LOAD}_{\mathcal{C}}$$

Solving this quadratic equation for Θ, we get $\Theta \geq \frac{-(L-2\,\Pi)+\sqrt{(L-2\,\Pi)^2+8L\,\Pi \times \mathrm{LOAD}_{\mathcal{C}}}}{4}$. $BO_{\mathcal{C}}(\Pi)$ is then given as

$$\begin{aligned} BO_{\mathcal{C}}(\Pi) =& \frac{\Theta}{\Pi} - \mathrm{LOAD}_{\mathcal{C}} \\ \geq& \frac{-(L - 2\,\Pi) + \sqrt{(L - 2\,\Pi)^2 + 8L\,\Pi\,\mathrm{LOAD}_{\mathcal{C}}}}{4\,\Pi} - \mathrm{LOAD}_{\mathcal{C}} \\ =& \frac{-A + \sqrt{A^2 + 16\,\Pi^2\,\mathrm{LOAD}_{\mathcal{C}}(1 - \mathrm{LOAD}_{\mathcal{C}})}}{4\,\Pi} \end{aligned} \tag{4.2}$$

We can show that for all Π, $\Pi \geq 1, BO'_{\mathcal{C}}(\Pi) \geq 0$, where $BO'_{\mathcal{C}}$ is the derivative of $BO_{\mathcal{C}}$ with respect to Π. Since resource periods are always greater than or equal to 1, using Equation (4.2) we get

$$BO_{\mathcal{C}}(1) \geq \frac{-(L - 2 + 4 \times \mathrm{LOAD}_{\mathcal{C}}) + \sqrt{(L - 2 + 4 \times \mathrm{LOAD}_{\mathcal{C}})^2 + 16 \times \mathrm{LOAD}_{\mathcal{C}}(1 - \mathrm{LOAD}_{\mathcal{C}})}}{4} \tag{4.3}$$

Any function of the form $\sqrt{N^2 + d}$ is equivalent to the following Taylor series expansion (see [70]).

$$\begin{aligned} \sqrt{N^2 + d} =& \sum_{i=0}^{\infty} \frac{(-1)^i (2i)! d^i}{(1 - 2i)(i!)^2 4^i N^{2i-1}} \\ \geq& N + d/2N - d^2/(8N^3) \end{aligned}$$

This series converges if $2N \geq d$. Let $N = (L-2+4\times \text{LOAD}_{\mathcal{C}})$ and $d = 16\times\text{LOAD}_{\mathcal{C}}(1-\text{LOAD}_{\mathcal{C}})$. Then, assuming $L \geq 2 + 4 \times \text{LOAD}_{\mathcal{C}}(1 - 2 \times \text{LOAD}_{\mathcal{C}})$ we get using Equation (4.3)

$$
\begin{aligned}
BO_{\mathcal{C}}(1) \geq & \frac{(-N + N + \frac{d}{2N} - \frac{d^2}{8N^3})}{4} \\
= & \frac{d}{8N}\left(1 - \frac{d}{4N^2}\right) \\
= & \frac{d}{8N}\left(1 - \frac{16 \times \text{LOAD}_{\mathcal{C}}(1 - \text{LOAD}_{\mathcal{C}})}{4N^2}\right) \\
= & \frac{d}{8N^3}(N^2 - 4 \times \text{LOAD}_{\mathcal{C}}(1 - \text{LOAD}_{\mathcal{C}})) \\
\geq & \frac{d}{8N^3}(N^2 - 1) && \text{LOAD}_{\mathcal{C}}(1 - \text{LOAD}_{\mathcal{C}}) \leq \frac{1}{4} \\
\geq & \frac{d}{16N} && 1 - \frac{1}{N^2} \geq \frac{1}{2} \\
= & \frac{16 \times \text{LOAD}_{\mathcal{C}}(1 - \text{LOAD}_{\mathcal{C}})}{16(L - 2 + 4 \times \text{LOAD}_{\mathcal{C}})} \\
\geq & \frac{\text{LOAD}_{\mathcal{C}}(1 - \text{LOAD}_{\mathcal{C}})}{L + 2}
\end{aligned}
$$

□

4.7.3 Comparison between INC and NINC

In this section we use the bandwidth upper bound of INC (Section 4.7.1) and the bandwidth lower bound of NINC (Section 4.7.2) to compare these two approaches. The following lemma shows that any non-elementary component in the hierarchical system under consideration, can have at most $n - (k - 1)$ components in its workload.

Lemma 4.6 *Consider a hierarchical system comprised of n elementary and k non-elementary components. Assuming any non-elementary component must have at least two components in its workload, the maximum number of components in any non-elementary component's workload is at most $n - (k - 1)$.*

Proof The underlying undirected graph of this hierarchical system is a rooted tree. Since there are exactly $n+k$ nodes in this tree, the number of edges is $n+k-1$. Furthermore, in the directed version of this tree, where there is an edge from a parent component to each of its children,

the total in-degree of all nodes is also $n + k - 1$. Now we use the fact that the total in-degree is equal to the total out-degree in a directed graph. Suppose a non-elementary component B has the maximum possible out-degree d, and every other non-elementary component has the minimum possible out-degree. Each elementary component has out-degree zero, and there are n such components. Each non-elementary component except B has out-degree two (assumption), and there are $k-1$ such components. Therefore, $n+k-1 = d+2(k-1)$, and therefore $d = n-k+1$. Therefore, the maximum number of components in any non-elementary component's workload is at most $n - k + 1$.

It is also easy to see that for every n and k, it is possible to construct a tree that has maximum out-degree $n - k + 1$. Suppose $k - 1$ of the non-elementary components have one other non-elementary component and one elementary component in their workloads. Then, the remaining $n - (k - 1)$ elementary components can all be assigned to the workload of last non-elementary component in the chain. This shows that our bound is tight. □

We now derive a bandwidth lower bound for the hierarchical system under consideration, when interfaces are generated using NINC.

Theorem 4.7 *For the hierarchical system under consideration, let B_{ninc} denote the bandwidth of its root component interface under* NINC. *Then,*

$$B_{ninc} \geq \sum_{i=1}^{n} B_i + \frac{Ak}{P^*} + \frac{\sum_{i=1}^{n} B_i(1 - \sum_{i=1}^{n} B_i)}{(n-k+1)P^* + 2}.$$

Proof Consider a non-elementary component whose workload comprises of periodic resource models generated using NINC. Let the total utilization ($\sum_i \frac{\Theta_i}{\Pi_i}$) of this workload be U and LCM denote the least common multiple of periods of resource models in this workload. Then, from

Theorem 4.5, a lower bound on the bandwidth of this component's interface under NINC is

$$U + \frac{U(1-U)}{\text{LCM}+2} \tag{4.4}$$

We make following two observations for this lower bound and the hierarchical system under consideration: (1) the lower bound is a non-decreasing function over U (first order derivative is positive), and (2) LCM for any non-elementary component is at most $(n-k+1)P^*$ (from Lemma 4.6).

Now consider only those non-elementary components whose workload comprises of at least one elementary component. If the workload comprises of only elementary components, then Equation (4.4) gives a lower bound on the bandwidth of its interface under NINC. On the other hand, if the workload has non-elementary components, let $\bar{U}$ denote the total utilization of all the elementary component interfaces in the workload. Then using observation (1) we get that $\bar{U} + \frac{\bar{U}(1-\bar{U})}{\text{LCM}+2}$ gives the bandwidth lower bound. Also, $\frac{Ak}{P^*}$ is the minimum total preemption overhead incurred by all the non-elementary components. Let j iterate over all non-elementary components whose workload comprises of only elementary components, and k iterate over all non-elementary components whose workload comprises of at least one but not all elementary components. Then,

$$\begin{aligned}
B_{ninc} \geq & \frac{Ak}{P^*} + \sum_j \left[U_j + \frac{U_j(1-U_j)}{\text{LCM}+2} \right] + \sum_k \left[\bar{U}_k + \frac{\bar{U}_k(1-\bar{U}_k)}{\text{LCM}+2} \right] \\
\geq & \frac{Ak}{P^*} + \sum_j U_j + \sum_k \bar{U}_k + \frac{(1-\sum_{i=1}^n B_i)}{\text{LCM}+2} \left[\sum_j U_j + \sum_k \bar{U}_k \right] && \sum_{i=1}^n B_i \geq U_j, \bar{U}_k \\
\geq & \frac{Ak}{P^*} + \left[1 + \frac{(1-\sum_{i=1}^n B_i)}{(n-k+1)P^*+2} \right] \left[\sum_j U_j + \sum_k \bar{U}_k \right] && \text{Observation (2)} \\
= & \frac{Ak}{P^*} + \sum_{i=1}^n B_i + \frac{\sum_{i=1}^n B_i (1-\sum_{i=1}^n B_i)}{(n-k+1)P^*+2} && \sum_j U_j + \sum_k \bar{U}_k = \sum_{i=1}^n B_i
\end{aligned}$$

□

The following corollary compares bandwidth upper bound of INC with the bandwidth lower bound of NINC.

Corollary 4.8

$$B_{inc} - B_{ninc} \leq A(n+k)\left(\frac{1}{\Pi_1} - \frac{1}{P^*}\right) - \frac{(B + \frac{nA}{P^*})(1 - B - \frac{nA}{\Pi_1})}{(n-k+1)P^* + 2}, \text{ where } B = \sum_{i=1}^{n} B_{i,\Pi_i}.$$

Proof Using Theorems 4.4 and 4.7 we get,

$$\begin{aligned}
B_{inc} - B_{ninc} \leq & \sum_{i=1}^{n} B_{i,\Pi_i} + \frac{A(n+k)}{\Pi_1} - \frac{Ak}{P^*} - \sum_{i=1}^{n} B_i - \frac{\sum_{i=1}^{n} B_i(1 - \sum_{i=1}^{n} B_i)}{(n-k+1)P^* + 2} \\
= & \frac{A(n+k)}{\Pi_1} - \frac{Ak}{P^*} - \sum_{i=1}^{n} \frac{A}{\Pi_i} - \frac{(B + \sum_{i=1}^{n} \frac{A}{\Pi_i})(1 - B - \sum_{i=1}^{n} \frac{A}{\Pi_i})}{(n-k+1)P^* + 2} \\
\leq & \frac{A(n+k)}{\Pi_1} - \frac{Ak}{P^*} - \sum_{i=1}^{n} \frac{A}{P^*} - \frac{(B + \sum_{i=1}^{n} \frac{A}{P^*})(1 - B - \sum_{i=1}^{n} \frac{A}{\Pi_1})}{(n-k+1)P^* + 2} \\
\leq & A(n+k)\left(\frac{1}{\Pi_1} - \frac{1}{P^*}\right) - \frac{(B + \frac{nA}{P^*})(1 - B - \frac{nA}{\Pi_1})}{(n-k+1)P^* + 2}
\end{aligned}$$

□

4.8 Conclusions and future work

In this chapter we developed incremental schedulability analysis techniques for hierarchical systems. These techniques support interface reuse, and can therefore be used in on-the-fly analysis like dynamic voltage scaling of processors and on-line admission tests for components. We developed multi-period interfaces that allow system designers to choose resource models, after taking into consideration different properties like resource bandwidth and preemption overhead. Finally, we also derived an upper bound for the resource overhead incurred by our technique, in comparison to an earlier proposed non-incremental approach based on periodic resource models.

Although we presented a general multi-period interface composition technique under EDF, identical period restriction is required to compose interfaces scheduled under DM. For wider applicability of our techniques, it is essential to support general composition under DM scheduler as well.

Transformation from resource models to periodic tasks, presented in Section 4.6.3, helps the parent scheduler to schedule underlying workload interfaces. Although these tasks collectively demand enough resource from the parent scheduler so that workload interfaces can be scheduled, they however do not specify how to distribute this available resource between workload interfaces. It is worth noting that the tasks themselves cannot be used for this purpose. Determining this resource distribution pattern is another interesting future work.

Chapter 5

Explicit deadline periodic (EDP) resource models

5.1 Introduction

Resource model based component interfaces and compositional schedulability analysis using them is a well studied technique (see [80, 97, 102, 101]). For example, in Chapter 2 we presented the periodic resource model which characterizes periodic resource allocation behaviors. In this chapter we introduce the *Explicit Deadline Periodic* (EDP) resource model, which generalizes periodic models with an explicit deadline parameter. An EDP resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$ repetitively provides Θ units of resource within Δ time units, where the period of repetition is Π. Note that an EDP model η is identical to a periodic resource model $\phi = \langle \Pi, \Theta \rangle$ whenever $\Delta = \Pi$. EDP resource models also characterize periodic resource allocation behaviors, and therefore can be transformed into periodic tasks. This enables their scheduling under many popular schedulers such as EDF and DM.

Resource models when viewed as component interfaces present a virtualization of the hardware platform to components. For each component, its interface represents a time-partitioned resource

supply of the uniprocessor platform. In this view, the bandwidth of a resource model defined as $\frac{\Theta}{\Pi}$ for both periodic and EDP, is a quantitative measure of the average amount of resource used by that model over time. Since there is only a finite resource bandwidth available from the platform (bandwidth of one), it is desirable to minimize this quantity when generating resource model based interfaces.

In comparison to the EDP model based analysis presented in this chapter, periodic model based techniques incur resource bandwidth overheads that can be explained as follows. Schedulability conditions for periodic models (Theorems 2.5 and 2.6 in Section 2.2.3) indicate that for a resource model to be able to schedule a component's workload, the blackout interval length of its sbf must be smaller than the earliest deadline in the workload. For a periodic model $\phi = \langle \Pi, \Theta \rangle$, sbf_ϕ is given by Definition 2.12 and the blackout interval is illustrated in Figure 2.7 in Section 2.2.1. It is easy to see that to decrease blackout interval of sbf_ϕ, either Θ must be increased or Π must be decreased. However, periods of resource model based interfaces are typically specified a priori by system designers; for instance to account for preemption overheads. Therefore, decrease in the blackout interval of sbf_ϕ necessarily requires an increase in Θ, *i.e.*, an increase in resource bandwidth. On the other hand, for an EDP resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$, we show that blackout interval length depends on both its capacity Θ and deadline Δ. Then, one may adjust the deadline parameter of an EDP model and decrease blackout interval, without increasing its resource bandwidth.

In addition to introducing EDP models, the contributions of this chapter include, (1) efficient algorithm to compute an optimal EDP model based interface for an elementary component (model with minimum resource bandwidth), and (2) exact transformation from EDP models to periodic tasks (this transforms a non-elementary component into an elementary component whose analysis can be done as in (1) above).

This chapter is organized as follows: In Section 5.2 we state our system model assumptions

and describe the problems that we address. In Section 5.3 we introduce EDP resource models, and in Section 5.4 we develop component interfaces based on them. We then develop transformations from EDP models to periodic tasks in Section 5.5. Finally, we conclude this chapter and discuss future work in Section 5.6.

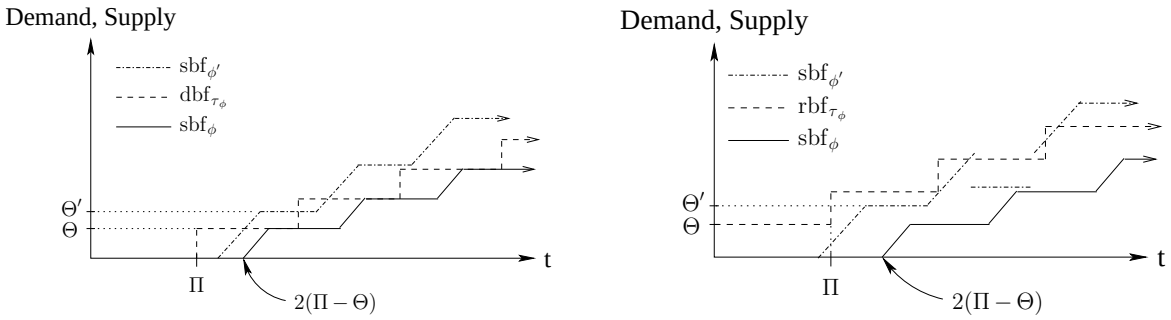
5.2 Problem statement

System assumptions. In this chapter we assume that tasks in the workload of a component are specified using the sporadic task model described in Section 2.1.1. Although we focus on one task model, techniques presented here can be applied to any task model whose dbf and rbf (Definitions 2.6 and 2.7) are computable.

Problem statement. In the introduction, we discussed how the relation between blackout interval and resource bandwidth, makes EDP models more suitable for compositional analysis than periodic resource models. In addition to this advantage, periodic model based techniques are undesirable for the following reasons:

- They use a linear lower bound of sbf_ϕ when generating component interfaces. This leads to resource overheads because corresponding schedulability conditions are sufficient, but not necessary.

- The transformation from resource model $\phi = \langle \Pi, \Theta \rangle$ to periodic task $\tau_\phi = (\Pi, \Theta, \Pi)$ also induces resource overheads. This can be explained by observing that the resource demand of task τ_ϕ is sufficient, but not necessary with respect to the supply guarantee of ϕ. In other words, any resource supply that can schedule τ_ϕ must supply at least as much resource as sbf_ϕ, but there can exist resource models R that satisfy sbf_ϕ and yet cannot schedule τ_ϕ. Figure 5.1(a) illustrates this shortcoming for an example in which periodic model $\phi' = \langle \Pi, \Theta' \rangle$, with $\text{sbf}_{\phi'}(\Pi) < \Theta$ and $\Theta' > \Theta$, cannot satisfy the demand of task τ_ϕ under



(a) under EDF (b) under DM

Figure 5.1: Sub-optimal transformations

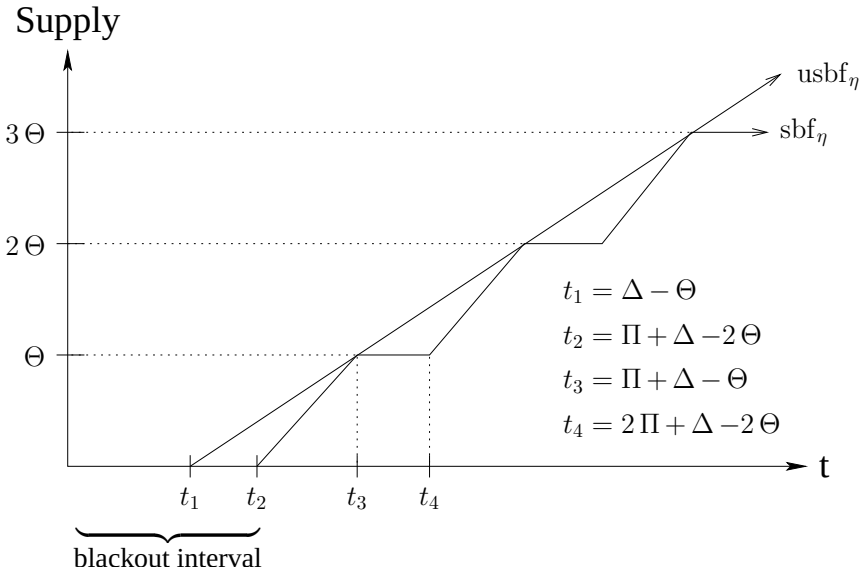


Figure 5.2: sbf and usbf of EDP resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$

EDF. Similarly, Figure 5.1(b) illustrates this shortcoming under DM.

In this chapter we develop EDP resource model based interfaces and compositional analysis techniques that overcome the aforementioned shortcomings.

5.3 Explicit deadline periodic resource model

An explicit deadline periodic (EDP) resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$ provides Θ units of resource within Δ time units, with this pattern repeating every Π time units. In this work we focus on EDP models with constrained deadlines, *i.e.*, $\Theta \leq \Delta \leq \Pi$. Recall that the supply bound function

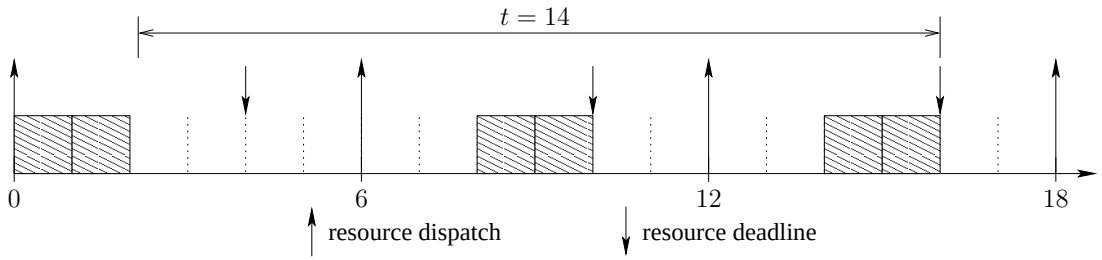


Figure 5.3: Resource supply of $\eta = \langle 6, 2, 4\rangle$ corresponding to $\text{sbf}_\eta(14)$

(sbf) of a resource model gives the smallest amount of resource that the model is guaranteed to provide in a time interval length. sbf of resource model η is given by Equation (5.1) and plotted in Figure 5.2. As shown in the figure, blackout interval length of sbf_η is $\Pi + \Delta - 2\,\Theta$. Thereafter the model provides Θ units of resource in every Π time units. Note that blackout interval of sbf_η can be smaller than the blackout interval $2\,\Pi - 2\,\Theta$ of a periodic model $\phi = \langle \Pi, \Theta\rangle$, even though η and ϕ have the same resource bandwidth. Corresponding to each sbf_η value, we can identify a resource supply pattern of η which generates that value. For example, the supply pattern of EDP model $\eta = \langle 6, 2, 4\rangle$ corresponding to $\text{sbf}_\eta(14)$ is shown in Figure 5.3. We also define a linear function usbf_η in Equation (5.2). This function, illustrated in Figure 5.2, is an upper bound of sbf_η and will be used to generate EDP model based interfaces.

$$\text{sbf}_\eta(t) = \begin{cases} \left\lfloor \frac{t-(\Delta-\Theta)}{\Pi} \right\rfloor \Theta + \max\left\{0, t - (\Pi + \Delta - 2\,\Theta) - \left\lfloor \frac{t-(\Delta-\Theta)}{\Pi} \right\rfloor \Pi\right\} & t \geq \Delta - \Theta \\ 0 & \text{Otherwise} \end{cases} \tag{5.1}$$

$$\text{usbf}_\eta(t) = \frac{\Theta}{\Pi}(t - (\Delta - \Theta)) \tag{5.2}$$

For elementary components, schedulability conditions under EDP resource models are identical to Theorems 2.5 and 2.6 in Section 2.2.2, assuming sbf_ϕ is replaced with sbf_η. We record these conditions in the following theorems.

Theorem 5.1 (Schedulability under EDF (EDP resource model)) *Let* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *denote a set of sporadic tasks, where for each* i, $\tau_i = (\text{T}_i, \text{C}_i, \text{D}_i)$. *Also, let* LCM

denote the least common multiple of minimum separations $\mathrm{T}_1, \ldots, \mathrm{T}_n$. *Component* $\mathcal{C} = \langle \mathcal{T}, \mathrm{EDF} \rangle$ *is schedulable using an EDP resource model* $\eta = \langle \Pi, \Theta, \Delta \rangle$ *iff* $\frac{\Theta}{\Pi} \geq U_{\mathcal{T}}$ *and*

$$\forall t \text{ s.t. } 0 < t \leq L, \mathrm{dbf}_{\mathcal{C}}(t) \leq \mathrm{sbf}_{\eta}(t), \tag{5.3}$$

where $L = \min\left\{\mathrm{LCM} + \max_{i=1,\ldots,n} \mathrm{D}_i, \frac{U_{\mathcal{T}}(\max_{i=1,\ldots,n}(\mathrm{T}_i - \mathrm{D}_i))}{1-U_{\mathcal{T}}}\right\}$. *Furthermore,* $\mathcal{C}$ *is* exactly schedulable *by* η *iff, in addition to the above condition,* $\exists t$ *s.t.* $\min_{i=1,\ldots,n} \mathrm{D}_i \leq t \leq L$ *and* $\mathrm{dbf}_{\mathcal{C}}(t) = \mathrm{sbf}_{\eta}(t)$.

Theorem 5.2 (Schedulability under DM (EDP resource model)) *Let* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *denote a set of sporadic tasks, where for each* i, $\tau_i = (\mathrm{T}_i, \mathrm{C}_i, \mathrm{D}_i)$. *Also, for all* $i < j$, *let* $\mathrm{D}_i \leq \mathrm{D}_j$. *Component* $\mathcal{C} = \langle \mathcal{T}, \mathrm{DM} \rangle$ *is schedulable using an EDP resource model* $\eta = \langle \Pi, \Theta, \Delta \rangle$ *iff*

$$\forall i : 1 \leq i \leq n, \exists t_i \in [0, \mathrm{D}_i] \text{ s.t. } \mathrm{rbf}_{\mathcal{C},i}(t_i) \leq \mathrm{sbf}_{\eta}(t_i) \tag{5.4}$$

Furthermore, $\mathcal{C}$ *is* exactly schedulable *by* η *iff, in addition to the above condition,* $\forall i : 1 \leq i \leq n, \forall t \in [0, \mathrm{D}_i], \mathrm{rbf}_{\mathcal{C},i}(t) \geq \mathrm{sbf}_{\eta}(t)$.

These schedulability conditions are agnostic to preemption related overheads incurred by task set $\mathcal{T}$. Since we use these conditions to generate EDP interfaces, we ignore preemption overheads in this chapter. We assume that period Π of EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$ is specified a priori by the designer, after taking into consideration preemption overheads. Then, the only unknown quantities in Equation (5.3) or (5.4) are capacity Θ and deadline Δ. Before we present our interface generation technique, we first establish a relationship between Θ and Δ that minimizes resource bandwidth. Specifically, we now define a *bandwidth optimal* EDP interface, and derive conditions on Θ and Δ under which this optimality is achieved.

Definition 5.1 (Bandwidth optimal EDP interface) *Let* $\mathcal{C} = \langle \mathcal{T}, \mathcal{S} \rangle$ *denote an elementary component. Given period value* Π, *an EDP resource model*

$\eta = \langle \Pi, \Theta, \Delta \rangle$ *is* bandwidth optimal *for component* $\mathcal{C}$*, iff its resource bandwidth is the minimum over all EDP models that have period* Π *and satisfy schedulability conditions for* $\mathcal{C}$*.*

The following theorem establishes a relationship between Θ and Δ such that resource model η is bandwidth optimal for component $\mathcal{C}$. Note that it uses the notion of exactly schedulable defined in Theorems 5.1 and 5.2.

Theorem 5.3 *Let* $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ *denote a set of sporadic tasks. Furthermore, let EDP model* $\eta = \langle \Pi, \Theta, \Delta \rangle$ *exactly schedule an elementary component* $\mathcal{C} = \langle \mathcal{T}, \mathcal{S} \rangle$*. Then,* η *is bandwidth optimal for* $\mathcal{C}$ *if* $\Delta = \Theta$*.*

Proof To prove this theorem we show that if any EDP model $\eta' = \langle \Pi, \Theta', \Delta' \rangle$ with $\Delta' > \Theta'$ exactly schedules component $\mathcal{C}$, then it must be the case that $\Theta' \geq \Theta$. We prove this by contradiction and assume $\Theta' < \Theta$. We first show that under this assumption $\mathrm{sbf}_{\eta'}(t) < \mathrm{sbf}_{\eta}(t)$, for all $t \geq \Delta' - \Theta'$. Since $\Theta' \leq \Delta' \leq \Pi$, $\frac{\Delta' - \Theta'}{\Pi} < 1$ and $\lfloor \frac{t}{\Pi} \rfloor$ can be at most one greater than $\left\lfloor \frac{t - \Delta' + \Theta'}{\Pi} \right\rfloor$. Hence, we consider the following two cases.

Case $\lfloor \frac{t}{\Pi} \rfloor = \left\lfloor \frac{t - \Delta' + \Theta'}{\Pi} \right\rfloor (= k)$**:**

In this case $k\,\Theta > k\,\Theta'$ and $\max\{0, t - (\Pi - \Theta) - k\,\Pi\} \geq \max\{0, t - (\Pi + \Delta' - 2\,\Theta') - k\,\Pi\}$. The second inequality holds because $\Pi - \Theta < \Pi + \Delta' - 2\,\Theta'$. Therefore, from Equation (5.1) we get

$$\begin{aligned}
\mathrm{sbf}_{\eta'}(t) &= \left\lfloor \frac{t - (\Delta' - \Theta')}{\Pi} \right\rfloor \Theta' + \max\left\{0, t - (\Pi + \Delta' - 2\,\Theta') - \left\lfloor \frac{t - (\Delta' - \Theta')}{\Pi} \right\rfloor \Pi\right\} \\
&= k\,\Theta' + \max\left\{0, t - (\Pi + \Delta' - 2\,\Theta') - k\,\Pi\right\} \\
&< k\,\Theta + \max\left\{0, t - (\Pi - \Theta) - k\,\Pi\right\} \\
&= \mathrm{sbf}_{\eta}(t)
\end{aligned}$$

Case $\lfloor \frac{t}{\Pi} \rfloor = k, \left\lfloor \frac{t - \Delta' + \Theta'}{\Pi} \right\rfloor = k - 1$**:**

Here $\mathrm{sbf}_{\eta'}(t) = (k-1)\,\Theta' + \max\{0, t - (\Pi + \Delta' - 2\,\Theta') - k\,\Pi + \Pi\}$. Observe that

$\max\{0, t - (\Pi + \Delta' - 2\,\Theta') - k\,\Pi + \Pi\} < \Theta'$ for all $t \geq \Delta' - \Theta'$. Therefore, $\text{sbf}_{\eta'}(t) < (k-1)\,\Theta' + \Theta' < k\,\Theta \leq \text{sbf}_{\eta}(t)$.

We now derive a contradiction by showing that resource model η' cannot schedule component $\mathcal{C}$. When $\mathcal{S} = \text{EDF}$, since η exactly schedules $\mathcal{C}$, from Theorem 5.1 we get $\text{sbf}_{\eta}(t') = \text{dbf}_{\mathcal{C}}(t')$ for some $t' \geq \min_{i=1,\ldots,n} \text{D}_i$. However, if $t' \geq \Delta' - \Theta'$, then we have shown that $\text{sbf}_{\eta'}(t') < \text{sbf}_{\eta}(t')$. Otherwise, $t' < \Delta' - \Theta'$ which implies $\text{sbf}_{\eta'}(t') = 0$. Since $\text{dbf}_{\mathcal{C}}(t') > 0$, we get that η' does not schedule component $\mathcal{C}$ in both these cases. When $\mathcal{S} = \text{DM}$, since η exactly schedules $\mathcal{C}$, from Theorem 5.2 we get $\forall i, \exists t' \in [0, \text{D}_i]$, such that $\text{rbf}_{\mathcal{C},i}(t') = \text{sbf}_{\eta}(t')$. Then using arguments similar as above, we can show that η' does not schedule component $\mathcal{C}$. Thus, we have derived a contradiction in all cases, and hence $\Theta' \geq \Theta$.

The result then follows, because any EDP model $\langle \Pi, \Theta'', \Delta' \rangle$ that schedules component $\mathcal{C}$, can be transformed into another EDP model $\langle \Pi, \Theta', \Delta' \rangle$ that exactly schedules $\mathcal{C}$ with $\Theta' \leq \Theta''$. □

5.4 EDP interfaces for elementary components

In this section, given an elementary component $\mathcal{C}$ and period value Π, we generate an EDP interface for $\mathcal{C}$. This interface satisfies the following notion of optimality.

Definition 5.2 (Bandwidth-deadline optimal EDP interface) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of sporadic tasks. $\eta^* = \langle \Pi, \Theta^*, \Delta^* \rangle$ is a* bandwidth-deadline optimal *EDP interface for component $\mathcal{C} = \langle \mathcal{T}, \mathcal{S} \rangle$ iff*

- *η^* is bandwidth optimal for $\mathcal{C}$, and*
- *For all EDP resource models $\eta = \langle \Pi, \Theta^*, \Delta \rangle$ such that η is bandwidth optimal for $\mathcal{C}$, $\Delta \leq \Delta^*$.*

Among all EDP interfaces that are bandwidth optimal for component $\mathcal{C}$, the interface which has the largest deadline is desirable. A larger deadline for the resource model implies a larger blackout

interval of its sbf. When transforming an EDP model to periodic task, demand of the generated task depends on sbf of the model. Then, a sbf with larger blackout interval implies delayed (and hence reduced) demand at the next level in scheduling hierarchy. In other words, a bandwidth-deadline optimal EDP interface not only minimizes resource bandwidth, but also reduces demand for the higher level parent scheduler. We now describe procedures to generate these interfaces when component $\mathcal{C}$ uses EDF or DM scheduler.

EDP interface under EDF. We assume period Π for EDP interface $\eta^* = \langle \Pi, \Theta^*, \Delta^* \rangle$ is given. The following procedure computes Θ^* and Δ^*.

1. Set $\Delta = \Theta$, evaluate Equation (5.3) for each time interval length, and choose the maximum value of $\Theta (= \Theta^*)$ over all interval lengths. We only need to consider those interval lengths at which $\text{dbf}_{\mathcal{C}}$ changes. Note that $\langle \Pi, \Theta^*, \Theta^* \rangle$ is bandwidth optimal for $\mathcal{C}$.

2. Set $\Theta = \Theta^*$, evaluate Equation (5.3) for each interval length, and choose the maximum value of $\Delta (= \Delta^*)$ over all interval lengths. Again, we only need to consider those interval lengths at which $\text{dbf}_{\mathcal{C}}$ changes. It is then easy to see that resource model η^* is bandwidth-deadline optimal for $\mathcal{C}$.

EDP interface under DM. We assume period Π for EDP interface $\eta^* = \langle \Pi, \Theta^*, \Delta^* \rangle$ is given. Procedure that generates η^* consists of the following two steps:

1. Set $\Delta = \Theta$, and evaluate Equation (5.4) for each task τ_i in component $\mathcal{C}$ and for each interval length. We only need to consider those interval lengths at which $\text{rbf}_{\mathcal{C},i}$ changes. Choose the minimum value of $\Theta (= \Theta_i^*)$ over all interval lengths and let $\Theta^* = \max_{i=1,\ldots,n} \Theta_i^*$. Resource model $\langle \Pi, \Theta^*, \Theta^* \rangle$ is bandwidth optimal for $\mathcal{C}$.

2. Set $\Theta = \Theta^*$, and evaluate Equation (5.4) for each task τ_i in $\mathcal{C}$ and for each interval length. Again, we only consider those interval lengths at which $\text{rbf}_{\mathcal{C},i}$ changes. Choose the maximum

value of $\Delta(=\Delta_i^*)$ over all interval lengths and let $\Delta^* = \min_{i=1,\ldots,n} \Delta_i^*$. Then, resource model η^* is bandwidth-deadline optimal for $\mathcal{C}$.

In order to generate EDP interfaces, we have to evaluate Equations (5.3) and (5.4) for different values of interval length. Since Θ and Δ appear inside floor functions in these equations, computing these values so that the equations are tight seems non-trivial. Therefore, we now present an algorithm to compute these values efficiently. Specifically, our algorithm computes the value of Θ (Θ_t^*) that satisfies Equation (5.3) or (5.4) for some interval length t, when corresponding dbf or rbf value D_t, period Π_t, and deadline Δ_t are all known. Note that although we only present an algorithm to compute Θ, a similar algorithm can be used to compute Δ given Θ.

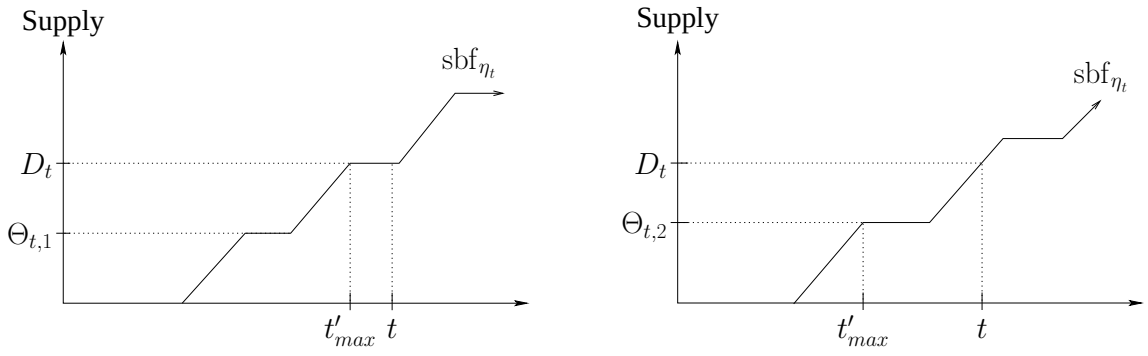
Algorithm 5 computes capacity Θ_t^* such that EDP model $\eta_t = \langle \Pi_t, \Theta_t, \Delta_t \rangle$, with $\Theta_t = \Theta_t^*$, satisfies the condition $\text{sbf}_{\eta_t}(t) = D_t$. This algorithm exploits the fact that there are only two possible values for the floor function in sbf_{η_t}, *i.e.*, for $t' = \left\lfloor \frac{t-\Delta_t+\Theta_t}{\Pi_t} \right\rfloor \Pi_t$. This follows from the observation that $\Theta_t \leq \Delta_t \leq \Pi_t$ and $\alpha < \Pi_t$, where α is the remainder obtained when t is divided by Π_t. For example, if $t = k\,\Pi_t + \alpha$, then t' is either $k\,\Pi_t$ or $(k-1)\,\Pi_t$. These two possible values of t' are computed in Lines 3,5 and 7 of the algorithm. In these computations, Θ_t is always set to the smallest possible value that can be assigned to it.

To compute Θ_t^* we then identify that value of t' for which $\text{sbf}_{\eta_t}(t') \leq D_t$ and $\text{sbf}_{\eta_t}(t')$ is the largest such value (t'_{max} in the algorithm). Then, the value of Θ_t corresponding to t'_{max} is such that Θ_t^* is greater than or equal to that value of Θ_t (set in Line 3,5, or 7). This follows from the fact that we always use smallest possible values for Θ_t when computing t'. Also, $t' = t'_{max}$ when Θ_t is set to Θ_t^*. This follows from the fact that as the value of Θ_t increases towards Θ_t^*, t'_{max} always satisfies properties mentioned above, *i.e.*, $\text{sbf}_{\eta_t}(t'_{max}) \leq D_t$ and $\text{sbf}_{\eta_t}(t'_{max})$ is the largest such value.

Algorithm 5 Algorithm to compute capacity Θ_t^* given interval length t

Input: t, D_t, Π_t, Δ_t
Output: Θ_t^*

Let $\eta_t = \langle \Pi_t, \Theta_t, \Delta_t \rangle$ denote an EDP resource model.
// We compute a value Θ_t^* for Θ_t, such that condition $\mathrm{sbf}_{\eta_t}(t) = D_t$ holds.
Let $t = k\,\Pi_t + \alpha$ and $t' = \left\lfloor \frac{t - \Delta_t + \Theta_t}{\Pi_t} \right\rfloor \Pi_t$.
// t' is the largest interval length $\leq t$, such that some rising portion of sbf_{η_t} ends at t'. $\mathrm{sbf}_{\eta_t}(t') = \mathrm{usbf}_{\eta_t}(t')$ at all such t'.
// First, compute different possible values for t' using smallest values of Θ_t that generate those t'.
Compute $s_0 = \mathrm{usbf}_{\eta_t}(t')$ with $\Theta_t = 0$, and let $t'_0 = \left\lfloor \frac{t - \Delta_t}{\Pi_t} \right\rfloor \Pi_t$.
if $\alpha < \Delta_t$ **then**
    Compute $s_1 = \mathrm{usbf}_{\eta_t}(t')$ with $\Theta_t = \Delta_t - \alpha$, and let $t'_1 = \left\lfloor \frac{t - \alpha}{\Pi_t} \right\rfloor \Pi_t$.
else
    Let $s_1 = s_0, t'_1 = t'_0$.
end if
// Now choose the largest value of $\mathrm{sbf}_{\eta_t}(t')$ and corresponding t' for which $\mathrm{sbf}_{\eta_t}(t') \leq D_t$.
if $s_1 > s_0$ and $s_1 \leq D_t$ **then**
    $t'_{max} \leftarrow t'_1$.
else
    $t'_{max} \leftarrow t'_0$.
end if
// Compute Θ_t^* assuming D_t is a multiple of Θ_t^*.
Compute $\Theta_t (= \Theta_{t,1})$ such that resource model η_t satisfies the condition $\mathrm{usbf}_{\eta_t}(t'_{max}) = D_t$.
// Compute Θ_t^* assuming D_t is not a multiple of Θ_t^*.
Compute $\Theta_t (= \Theta_{t,2})$ such that resource model η_t satisfies the condition $\mathrm{usbf}_{\eta_t}(t'_{max} + \Pi_t) = D_t + t'_{max} + \Pi_t - t$.
// Choose the smallest, real-valued capacity as Θ_t^*.
return $\Theta_t^* = \min_i \Theta_{t,i}$, where $\Theta_{t,i} \in \mathbb{R}^+$ and $i \in \{1, 2\}$.



(a) D_t is a multiple of Θ_t^* (b) D_t is not a multiple of Θ_t^*

Figure 5.4: Illustrations for Algorithm 5

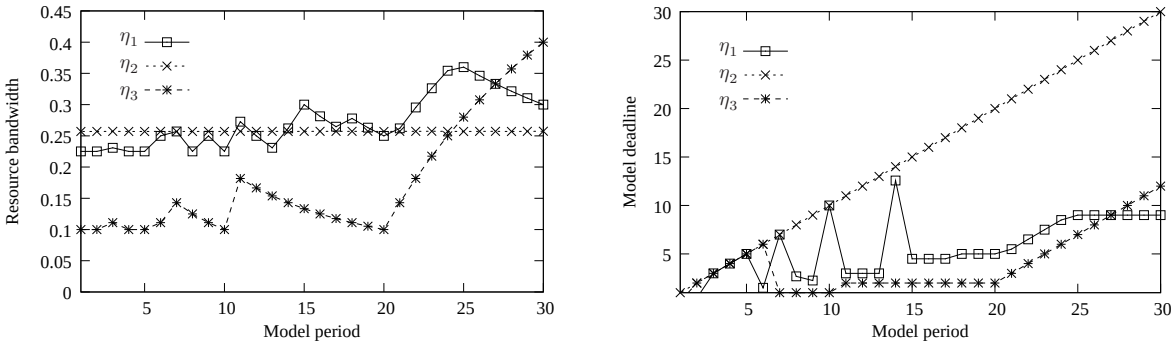
In Lines 14 and 15 we compute Θ_t^* for the two different cases shown in Figure 5.4 (D_t is either a multiple of Θ_t^* or not). These figures are only for illustration purposes, and give intuition for the two cases that we consider. If D_t is a multiple of Θ_t^*, then we use the condition $\text{usbf}_{\eta_t}(t'_{max}) = D_t$ to compute Θ_t^* (Figure 5.4(a)). Otherwise, we use the condition $\text{usbf}_{\eta_t}(t'_{max} + \Pi_t) = D_t + t'_{max} + \Pi_t - t$ (Figure 5.4(b)). Since $\text{sbf}_{\eta_t}(t')$ is a multiple of Θ_t for all values of t', $\text{sbf}_{\eta_t}(t'_{max}) = \text{usbf}_{\eta_t}(t'_{max})$ and $\text{sbf}_{\eta_t}(t'_{max} + \Pi_t) = \text{usbf}_{\eta_t}(t'_{max} + \Pi_t)$. Then, from the figures it is easy to see that $\text{sbf}_{\eta_t}(t) = D_t$ when Θ_t is set to Θ_t^*. Furthermore, since usbf_{η_t} is a quadratic function of Θ_t, these computations can be done in constant time.

Algorithm 5 can be used to compute the bandwidth optimal EDP interface $\langle \Pi, \Theta^*, \Theta^* \rangle$. A similar algorithm, based on different possible values of the floor function, can be used to compute the bandwidth-deadline optimal EDP interface $\langle \Pi, \Theta^*, \Delta^* \rangle$. Since each invocation of Algorithm 5 runs in constant time, the interface generation procedures described earlier have pseudo-polynomial running time. As an aside, since periodic resource models are special instances of EDP models, Algorithm 5 can also be used in periodic resource model based analysis techniques.

Example 5.1 *Consider components $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ shown in Figure 2.4. Let the workload of these components be comprised of sporadic tasks as shown in Table 5.1. We computed EDP interfaces η_1, η_2, and η_3 for components $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ respectively, using techniques described in this section.*

Component	Workload	Load: $\max_t \left\{ \frac{\text{dbf}(t)}{t} \right\}$
$\mathcal{C}_1$	$\{(45, 2, 25), (65, 3, 30), (85, 4, 40)\}$	0.225
$\mathcal{C}_2$	$\{(35000, 2000, 25000), (55000, 3000, 55000), (75000, 4000, 25000)\}$	0.24
$\mathcal{C}_3$	$\{(45, 1, 45), (75, 2, 20)\}$	0.1

Table 5.1: Workload of components $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$



(a) Bandwidth Vs. Period

(b) Deadline Vs. Period

Figure 5.5: EDP interfaces η_1, η_2, and η_3

The resource bandwidth and deadlines of these interfaces are plotted in Figure 5.5. From these plots we can observe that there are period values for which resource bandwidth of interface is equal to the corresponding component load. This means that for these period values our interfaces do not incur any resource overhead. From these plots we can also see that as period increases, resource bandwidth for interfaces also gradually increase. Furthermore, from Figure 5.5(a) it can be seen that the bandwidth has local peaks and troughs. It is then advantageous to choose a period value such that the corresponding bandwidth is a trough. Similarly, since the deadline also has local perturbations (Figure 5.5(b)), it is desirable to choose a period value such that the corresponding deadline is a peak.

5.5 EDP interfaces for non-elementary components

In this section we develop transformations from EDP resource models to periodic tasks, so that workloads of non-elementary components can be mapped to periodic task systems. Analysis of such transformed components can then be performed using techniques described in the previous section. For the periodic tasks that we generate, EDF and DM are optimal dynamic- and static-priority schedulers respectively. We also show that these transformations are exact in that they are both necessary and sufficient for schedulability.

5.5.1 Interface transformation under EDF scheduler

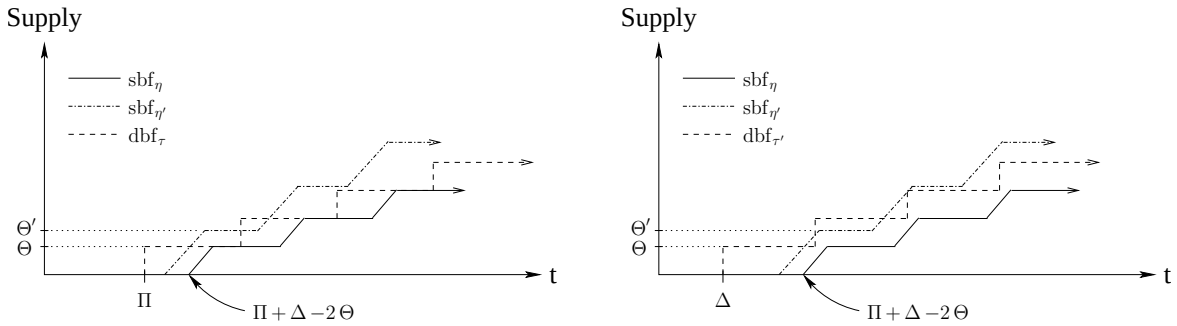
Consider a component $\mathcal{C}$ with EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$, such that $\mathcal{C}$ belongs to the workload of a higher level component that uses EDF. We now present a function $\mathcal{RT}_{\text{EDF}}$ that converts interface η into a periodic task τ. This task is *demand-supply optimal* which we define as follows.

Definition 5.3 (Demand-supply optimal task under EDF) *A periodic task τ scheduled under* EDF *is* demand-supply optimal *for an EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$ iff*

- *For any resource model R that can supply sufficient resource to satisfy the demand of task τ, $\text{sbf}_R \geq \text{sbf}_\eta$, and*
- *For any resource model R that need not always supply enough resource to satisfy the demand of task τ, $\exists t$ such that $\text{sbf}_R(t) < \text{sbf}_\eta(t)$.*

Note that the EDF scheduler we consider here is the scheduler at the next level in scheduling hierarchy. For example, to transform the interface of component $\mathcal{C}_1$ shown in Figure 2.4, relevant scheduler is the one used by component $\mathcal{C}_4$.

Consider simple transformations that generate periodic task $\tau = (\Pi, \Theta, \Pi)$ or $\tau' = (\Pi, \Theta, \Delta)$ for the EDP interface η. In general, these tasks need not be demand-supply optimal. As shown



(a) Periodic task $\tau = (\Pi, \Theta, \Pi)$ (b) Periodic task $\tau' = (\Pi, \Theta, \Delta)$

Figure 5.6: Sub-optimal transformations for EDP interfaces

in Figure 5.6, when $\Delta > \Theta$, there exists an EDP resource model $\eta' = \langle \Pi, \Theta', \Delta' \rangle$ such that $\text{sbf}_{\eta'} \geq \text{sbf}_\eta$, but η' cannot satisfy the resource requirements of τ or τ'. Hence, we define our transformation $\mathcal{RT}_{\text{EDF}}$ as follows.

Definition 5.4 *Given an EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$, transformation $\mathcal{RT}_{\text{EDF}}$ from model η to a periodic task is defined as $\mathcal{RT}_{\text{EDF}}(\eta) = (\Pi, \Theta, \Pi + \Delta - \Theta)$.*

Consider a non-elementary component $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \text{EDF} \rangle$. For each component $\mathcal{C}_i$, let η_i denote its EDP interface. Then, EDP interface for $\mathcal{C}$ can be generated by (1) transforming each interface η_i into a periodic task τ_i using Definition 5.4, and (2) generating an EDP interface for $\mathcal{C}$ using techniques described in Section 5.4. We now prove that transformation $\mathcal{RT}_{\text{EDF}}$ generates demand-supply optimal tasks.

Theorem 5.4 *Periodic task $\tau_\eta = (\Pi, \Theta, \Pi + \Delta - \Theta)$ generated by $\mathcal{RT}_{\text{EDF}}$ is demand-supply optimal for EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$.*

Proof We first prove that the transformation is sufficient for schedulability, *i.e.*, we show for any resource model R that can satisfy the resource requirements of τ_η, $\text{sbf}_R \geq \text{sbf}_\eta$. We consider two different types of interval lengths for this purpose: one where the length is in the range $(n\,\Pi + \Delta - 2\,\Theta, n\,\Pi + \Delta - \Theta)$, and the other where the length is in the range $[n\,\Pi + \Delta - \Theta, (n +$

$1)\,\Pi + \Delta - 2\,\Theta]$, for some $n \geq 1$. Informally, these intervals correspond to the rising and flat portions of sbf_η, respectively.

Case 1: $t \in (n\,\Pi + \Delta - 2\,\Theta, n\,\Pi + \Delta - \Theta)$

We prove this case by contradiction. Let $\text{sbf}_R(t) < \text{sbf}_\eta(t)$. For interval length $t' = n\,\Pi + \Delta - \Theta$ we have $\text{sbf}_R(t') \geq \text{dbf}_{\tau_\eta}(t')$ because resource model R satisfies the demand of task τ_η. Also, $\text{sbf}_\eta(t') = \text{dbf}_{\tau_\eta}(t')$ by definition. These together imply $\text{sbf}_\eta(t') - \text{sbf}_\eta(t) < \text{sbf}_R(t') - \text{sbf}_R(t)$. This is a contradiction, because in interval $[t, t']$ sbf_η is always rising with unit slope, and no sbf on an uniprocessor platform can rise with greater slope. Therefore, $\text{sbf}_R(t) \geq \text{sbf}_\eta(t)$.

Case 2: $t \in [n\,\Pi + \Delta - \Theta, (n+1)\,\Pi + \Delta - 2\,\Theta]$

In this case $\text{dbf}_{\tau_\eta}(t) = \text{sbf}_\eta(t)$. Also, any resource model R that supplies enough resource to satisfy the demand of τ_η must satisfy the condition $\text{sbf}_R(t) \geq \text{dbf}_{\tau_\eta}(t)$. Hence, $\text{sbf}_R(t) \geq \text{sbf}_\eta(t)$ in this case as well.

Combining Cases 1 and 2 above we get that $\text{sbf}_R(t) \geq \text{sbf}_\eta(t)$ for all $t \geq \Pi + \Delta - 2\,\Theta$. Since $\text{sbf}_\eta(t) = 0$ for all $t < \Pi + \Delta - 2\,\Theta$, we have proved that the amount of resource required to schedule τ_η is at least as much as sbf_η.

To prove that the transformation is necessary for schedulability, we observe that $\text{dbf}_{\tau_\eta} \leq \text{sbf}_\eta$. Hence, any resource model R such that $\text{sbf}_R \geq \text{sbf}_\eta$ can supply enough resource to satisfy the demand of τ_η. □

Discussion. Although the worst-case resource demand of periodic task τ_η is sufficient to satisfy sbf_η (Theorem 5.4), task τ_η itself cannot be directly used for distributing resource from the parent component to interface η. Instead, the following dynamic job dispatch scheme can be used. A job of task τ_η is dispatched for execution at the same time instant as the relative finish time of previous job, if the finish time is smaller than Π. Otherwise, the job is dispatched at its regular, relative dispatch time Π. Since the parent component's interface supplies resource

satisfying dbf_{τ_η}, it always guarantees at least Θ resource units in any time interval of length $\Pi + \Delta - \Theta$. Therefore, the parent interface can schedule the dynamic job dispatches described above.

5.5.2 Interface transformation under DM scheduler

In this section we present a function that transforms EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$ scheduled under DM, into a demand-supply optimal periodic task τ_η. Consider the transformation $\mathcal{RT}_{\text{EDF}}$ presented in the previous section. It is not a good transformation under DM, because (1) DM is not an optimal fixed-priority scheduler for tasks with deadline greater than period [75], and (2) although Lehoczky [75] has given a response time based schedulability condition for such systems over a dedicated resource, we do not have any schedulability condition over a partitioned resource such as EDP. Other simple transformations to consider are the ones that generate periodic task (Π, Θ, Π) or (Π, Θ, Δ). Examples in Figure 5.6 show that these transformations incur resource overhead under DM as well.

We now develop a transformation $\mathcal{RT}_{\text{DM}}$ from EDP interfaces to constrained deadline periodic tasks. This transformation depends on the period of component interface at the next level in scheduling hierarchy. For example, transformation of component $\mathcal{C}_3$'s interface in the hierarchical system shown in Figure 2.4, depends on the interface period of component $\mathcal{C}_5$. Since interface periods are specified a priori by the designer, this dependence is not an issue. Let Π' denote this period value for the parent component's interface. To define $\mathcal{RT}_{\text{DM}}$, we first compute a *bandwidth optimal resource supply* $\eta^* = \langle \Pi', \Theta^*, \Theta^* \rangle$ for interface $\eta = \langle \Pi, \Theta, \Delta \rangle$. This resource supply is defined as follows.

Definition 5.5 (Bandwidth optimal resource supply) *Given period* Π' *and EDP interface*

$\eta = \langle \Pi, \Theta, \Delta \rangle$, *resource model* $\eta^* = \langle \Pi', \Theta^*, \Theta^* \rangle$ *is a* bandwidth optimal resource supply *for* η *iff*

$$\forall t > 0, \mathrm{sbf}_{\eta^*}(t) \geq \mathrm{sbf}_{\eta}(t) \qquad \textit{(sufficiency)} \tag{5.5}$$

$$(\exists t > 0, \mathrm{sbf}_{\eta^*}(t) = \mathrm{sbf}_{\eta}(t)) \vee \left(\frac{\Theta^*}{\Pi'} = \frac{\Theta}{\Pi}\right) \qquad \textit{(necessity)} \tag{5.6}$$

These conditions imply that any EDP resource model with period Π' must have capacity at least Θ^* to provide supply as much as interface η (note that η^* is bandwidth optimal as per Theorem 5.3). We use η^* to generate the demand-supply optimal periodic task τ_η. This notion of optimality is similar to the EDF case, except that it depends on period Π'.

Definition 5.6 (Demand-supply optimal task under DM) *Let* $\eta = \langle \Pi, \Theta, \Delta \rangle$ *denote an EDP interface scheduled under* DM*, and* Π' *denote period of component interface at the next level from* η *in the hierarchical system. Then, a periodic task* τ *is demand-supply optimal for interface* η *iff*

- *For any EDP resource model* η' *with period* Π'*, that can supply sufficient resource to satisfy the demand of* τ*,* $\mathrm{sbf}_{\eta'} \geq \mathrm{sbf}_{\eta}$*, and*

- *For any EDP resource model* η' *with period* Π'*, that need not always supply enough resource to satisfy the demand of* τ*,* $\exists t$ *such that* $\mathrm{sbf}_{\eta'}(t) < \mathrm{sbf}_{\eta}(t)$*.*

In the rest of this section we define capacity Θ^* corresponding to resource model η^*, and present our transformation $\mathcal{RT}_{\mathrm{DM}}$ using η^*.

Theorem 5.5 *Given EDP interface* $\eta = \langle \Pi, \Theta, \Delta \rangle$ *and period* Π'*, the bandwidth optimal resource*

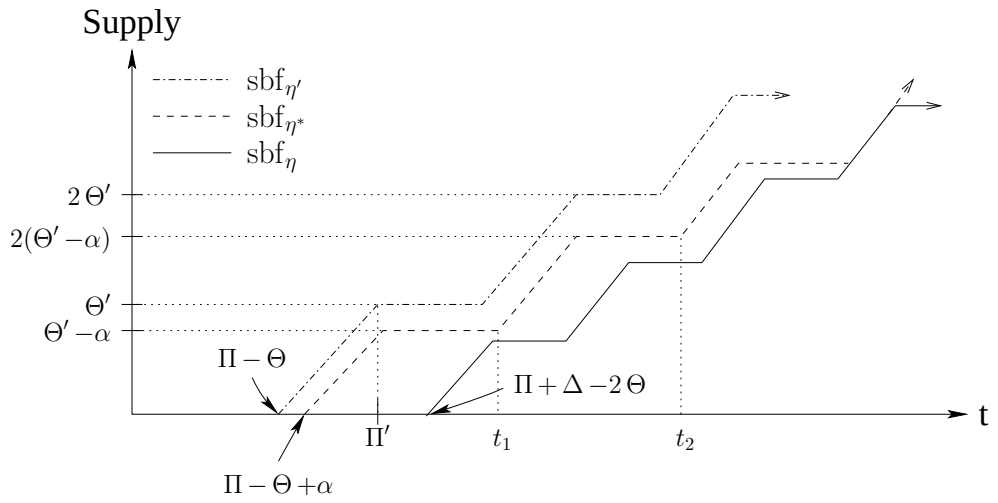


Figure 5.7: Illustration for Case 1 of Theorem 5.5

supply $\eta^* = \langle \Pi', \Theta^*, \Theta^* \rangle$ *is such that*

$$\Theta^* = \begin{cases} \max\left\{\Theta + \frac{k\,\Theta}{\Delta}, k - \Delta + 2\,\Theta\right\} & \Pi' = \Pi + k, k \geq 0 \\ \frac{\Theta}{k} & \Pi' = \frac{\Pi}{k}, k \geq 2 \\ \Theta_C & \Pi' = \frac{\Pi}{k} + \beta, k \geq 2 \end{cases}$$

where $\beta \in \left(0, \frac{\Pi}{k(k-1)}\right)$ *and* k *is an integer. If there exists* $n \in \left(\frac{\Delta-\Theta}{k\beta}, \frac{\Delta-\Theta}{k\beta} + \Pi' + 1\right)$ *such that* $nk\beta - (\Delta - \Theta) = l\,\Pi' + \gamma$ *and* n *is the smallest integer satisfying* $\gamma \geq \frac{n\,\Theta+\gamma}{nk-l}$*, then* $\Theta_C = \frac{n\,\Theta}{nk-(l+1)}$*. Otherwise,* $\Theta_C = \frac{\Theta}{k} + \frac{\beta\,\Theta}{\Pi}$*.*

Proof We separately consider the two cases $\Pi' \geq \Pi$ and $\Pi' < \Pi$.

Case 1: $\Pi' = \Pi + k, k \geq 0$

Consider the sbf of resource model $\eta' = \langle \Pi', \Theta', \Theta' \rangle$ shown in Figure 5.7, where $\Theta' = \Theta + k$. Then $\text{sbf}_{\eta'}(\Pi) = \Theta$, $\Pi' - \Theta' = \Pi - \Theta$, and $\frac{\Theta'}{\Pi'} \geq \frac{\Theta}{\Pi}$. Therefore, resource model η' is a bandwidth optimal resource supply for resource model $\langle \Pi, \Theta, \Theta \rangle$ (from Definition 5.5). Furthermore, η' also satisfies the sufficiency condition for resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$ in Definition 5.5. Hence, $\Theta^* \leq \Theta + k$.

We now compute a non-negative quantity α such that $\Theta^* = \Theta' - \alpha = \Theta + k - \alpha$. Consider the EDP resource model $\eta^* = \langle \Pi', \Theta' - \alpha, \Theta' - \alpha \rangle$ as shown in Figure 5.7. Observe that $\alpha = 0$ if and

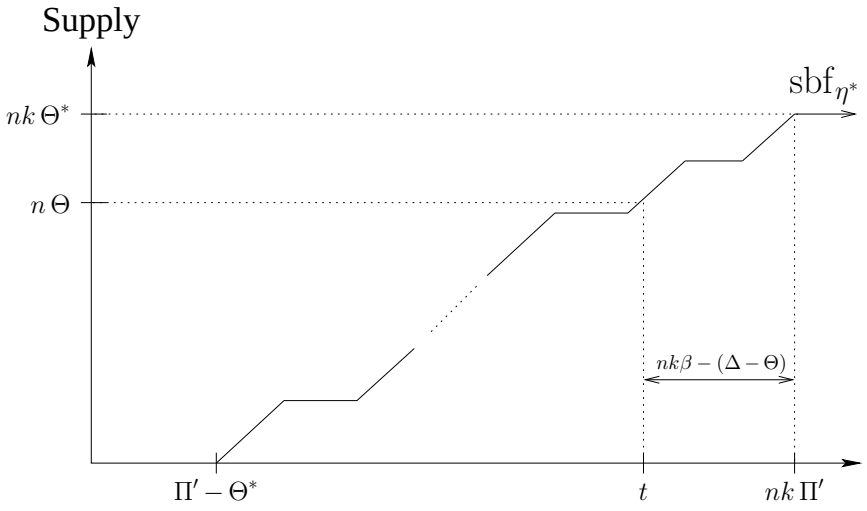


Figure 5.8: Illustration for Case 2 of Theorem 5.5

only if $\Delta = \Theta$, and in this case the proof is immediate. Therefore, in the remainder of this proof we assume $\alpha > 0$. We further assume that $\alpha < \Delta - \Theta \leq \Pi - \Theta$, and separately handle the case when $\alpha = \Delta - \Theta$.

Now, let $t_1 = \Pi' \left\lfloor \frac{n\,\Theta}{\Theta^*} \right\rfloor + \Pi' - \Theta^*$ and $t_2 = \Pi' \left\lceil \frac{n\,\Theta}{\Theta^*} \right\rceil + \Pi' - \Theta^*$ be any two interval lengths for which $n \in \{\mathbb{N} \bigcup \{0\}\}$ is unique (illustrated in Figure 5.7 for $n = 2$). Informally, sbf_η has exactly one flat segment between t_1 and t_2. Since $\Pi' - \Theta^* > \Pi - \Theta$, $t_2 - t_1 > \Pi$. Then, it can never be the case that $\text{sbf}_{\eta^*}(t_1) = \text{sbf}_\eta(t_1)$, and sbf_{η^*} at t_1 touches a rising segment of sbf_η. If this happens, then $\text{sbf}_{\eta^*}(t_2) < \text{sbf}_\eta(t_2)$ and η^* is not a bandwidth optimal resource supply for η. In fact, we can show that if $\text{sbf}_{\eta^*}(t) = \text{sbf}_\eta(t)$ with sbf_{η^*} at t touching a rising segment of sbf_η, then (1) $\left\lfloor \frac{n\,\Theta}{\Theta^*} \right\rfloor \Pi' + \Pi' - \Theta^* = \left\lfloor \frac{m\,\Theta}{\Theta^*} \right\rfloor \Pi' + \Pi' - \Theta^* (= t)$ for some $m \neq n$, where $m, n \in \{\mathbb{N} \bigcup \{0\}\}$, and (2) $t - \Pi'$ and t have properties identical to t_1 and t_2 respectively. This follows from the assumption that $0 < \alpha < \Delta - \Theta$. Likewise, we can also show that if $\text{sbf}_{\eta^*}(t) = \text{sbf}_\eta(t)$ with sbf_{η^*} at t touching a flat segment of sbf_η, then either t is as above, or t has properties that are identical to t_1. Let l_t denote the index, starting from 0, of all such interval lengths t sorted in an increasing sequence.

Then, the following conditions hold for each such t.

$$\left(\left\lfloor \frac{t}{\Pi'} \right\rfloor + 1\right)\Theta^* \geq \left(\left\lfloor \frac{t}{\Pi'} \right\rfloor + 2 + l_t\right)\Theta$$
$$l_t(\Pi - \Theta) + (\Delta - \Theta) \geq \left(\left\lfloor \frac{t}{\Pi'} \right\rfloor + 1\right)\alpha$$

Informally, first inequality represents the vertical gap between $\text{sbf}_{\eta^*}(t + \Pi')$ and the preceding horizontal segment of sbf_η (case 1 above), and second inequality represents the horizontal gap at $\text{sbf}_{\eta^*}(t)$ between t and the succeeding rising segment of sbf_η (case 2 above). Combining these conditions we get

$$\alpha \leq k\left(1 - \frac{(l_t + 1)\,\Theta}{l_t\,\Pi + \Delta}\right)$$

Since this condition must hold for all such t,

$$\alpha \leq \min_{l_t}\left\{k\left(1 - \frac{(l_t + 1)\,\Theta}{l_t\,\Pi + \Delta}\right)\right\}$$

Observe that the RHS here is minimized when $l_t = 0$, because $\Theta \leq \Delta \leq \Pi$. Furthermore, observe that α can never be greater than $\Delta - \Theta$. Therefore,

$$\begin{aligned}\Theta^* &= \Theta' - \min\left\{k\left(1 - \frac{\Theta}{\Delta}\right), \Delta - \Theta\right\} \\ &= \max\left\{\Theta + \frac{k\,\Theta}{\Delta}, k - \Delta + 2\,\Theta\right\}\end{aligned}$$

Case 2: $\Pi' = \frac{\Pi}{k} + \beta, k \geq 2, \beta \in \left[0, \frac{\Pi}{k(k-1)}\right)$

Since resource model η^* is a bandwidth optimal resource supply for interface η, $\frac{\Theta^*}{\Pi'} \geq \frac{\Theta}{\Pi}$ (by definition). Therefore, $\Theta^* \geq \frac{\Theta}{k} + \frac{\beta\,\Theta}{\Pi}$. Also, $\forall n \geq 1$ s.t. $t = n\,\Pi + \Delta - \Theta$, $\text{sbf}_{\eta^*}(t) \geq n\,\Theta$. If this condition is satisfied, then the sufficiency criteria of bandwidth optimal resource supply is met (Equation (5.5)). Furthermore, if this condition is tight for some n, then we also get necessity (Equation (5.6)). We use these observations to compute Θ^*.

If $\Theta^* \geq \frac{\Theta}{k}$, then $\text{sbf}_{\eta^*}(nk\,\Pi') = nk\,\Theta^* \geq n\,\Theta$. Since $n\,\Pi + \Delta - \Theta = nk\,\Pi' - (nk\beta - (\Delta - \Theta))$, $\forall n$ s.t. $nk\beta - (\Delta - \Theta) \leq 0$, we get $\text{sbf}_{\eta^*}(t) \geq n\,\Theta$. Therefore, if $n \leq \frac{\Delta - \Theta}{k\beta}$ or $\beta = 0$, $\Theta^* \geq \frac{\Theta}{k}$ is also sufficient.

We now consider the case when $n > \frac{\Delta - \Theta}{k\beta}$ and $\beta \neq 0$. Here t is smaller than $nk\,\Pi'$ by $nk\beta - (\Delta - \Theta)$, and we require that $\text{sbf}_{\eta^*}(t) \geq n\,\Theta$. In other words, the total resource supply of η^* in an interval of length $nk\beta - (\Delta - \Theta)$ immediately preceding time instant $nk\,\Pi'$, must be at most $nk\,\Theta^* - n\,\Theta$. This interval of interest is shown in Figure 5.8. Assuming $nk\beta - (\Delta - \Theta) = l\,\Pi' + \gamma$, we then require

$$\forall n, nk\,\Theta^* - n\,\Theta \geq l\,\Theta^* + \min\{\Theta^*, \gamma\}$$

$$\Rightarrow \Theta^* \geq \max_n \left\{ \min \left\{ \frac{n\,\Theta}{nk - (l+1)}, \frac{n\,\Theta + \gamma}{nk - l} \right\} \right\}$$

Consider a n for which $\gamma < \frac{n\,\Theta + \gamma}{nk - l} (= \Theta^*)$. Since $\Theta^* \leq \Theta$, we then get $\frac{\Pi'\,\Theta}{\Pi} \geq \frac{n\,\Theta + \gamma}{nk - l}$. Hence, we can ignore the term $\frac{n\,\Theta + \gamma}{nk - l}$ while computing Θ^*. Then, $\Theta^* = \frac{n\,\Theta}{nk - (l+1)}$ for the smallest n which satisfies $\gamma \geq \frac{n\,\Theta + \gamma}{nk - l}$ ($\frac{n\,\Theta}{nk - (l+1)}$ decreases with increasing n). This n, if it exists, lies in $\left(\frac{\Delta - \Theta}{k\beta}, \frac{\Delta - \Theta}{k\beta} + \Pi' + 1 \right)$, because γ is the remainder of $nk\beta - (\Delta - \Theta)$ divided by Π'. □

Transformation $\mathcal{RT}_{\text{DM}}$ can then be defined as follows.

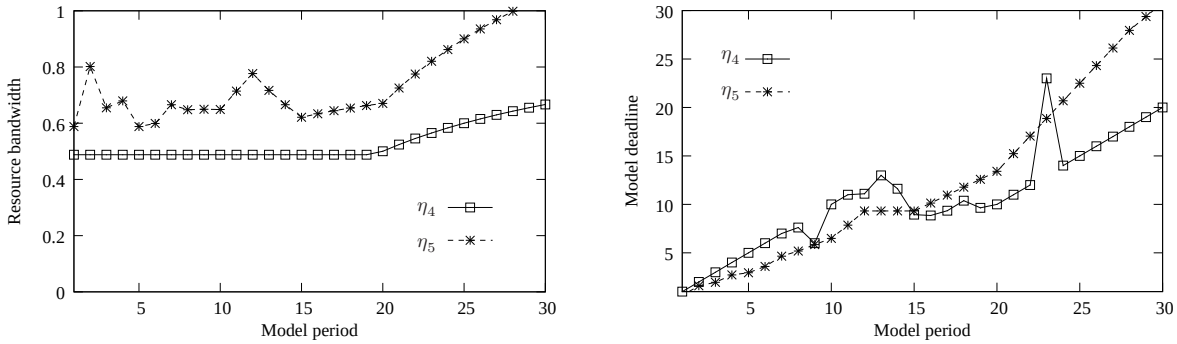
Definition 5.7 *Given an EDP interface $\eta = \langle \Pi, \Theta, \Delta \rangle$ and period value Π', transformation $\mathcal{RT}_{\text{DM}}$ from η to a periodic task is defined as $\mathcal{RT}_{\text{DM}}(\eta, \Pi') = (\Pi', \Theta^*, \Pi')$, where Θ^* is given by Theorem 5.5.*

Consider a non-elementary component $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \text{DM} \rangle$. For each component $\mathcal{C}_i$, let η_i denote its EDP interface, and Π' denote interface period for component $\mathcal{C}$. Then, EDP interface for component $\mathcal{C}$ can be generated by: (1) transforming each interface η_i into a periodic task τ_i using Definition 5.7, and (2) generating an EDP interface for $\mathcal{C}$ using techniques described in Section 5.4. Observe that $\mathcal{RT}_{\text{DM}}$ transforms interfaces $\eta_1, \ldots, \eta_n$ into periodic tasks that have the same period and deadline (Π'). The following corollary of Theorem 5.5 is a direct consequence of this observation.

Corollary 5.6 *Transformation $\mathcal{RT}_{\text{DM}}$ given in Definition 5.7, generates a demand-supply optimal periodic task for interface η under* DM.

Discussion. Similar to the EDF case, periodic task in Definition 5.7 cannot be directly used for distributing resource from the parent component to interface η. Instead, we can use the same dynamic job dispatch scheme that was described at the end of Section 5.5.1. Since, in this case as well, the parent interface provides resource based on the dbf of the task in Definition 5.7, it can satisfy the resource demand of these dynamic job dispatches.

Example 5.2 *We now demonstrate our analysis techniques on components $\mathcal{C}_4$ and $\mathcal{C}_5$ shown in Figure 2.4. Let the periods of chosen EDP interfaces η_1, η_2, and η_3 be* 13, 27, *and* 20 *respectively,* i.e., *let $\eta_1 = \langle 13, 3, 3\rangle, \eta_2 = \langle 27, 6.95, 27\rangle$, and $\eta_3 = \langle 20, 2, 2\rangle$ (these values are obtained from Figure 5.5). Now interfaces η_1 and η_2 are scheduled under* EDF *in component $\mathcal{C}_4$. Hence, from Definition 5.4 we get that the periodic tasks that form the workload of $\mathcal{C}_4$ are $(13, 3, 13)$ and $(27, 6.95, 47.05)$. EDP interface η_4 for component $\mathcal{C}_4$ is then plotted in Figure 5.9. In this figure we have plotted the resource bandwidth and deadline of η_4 for varying period values. We then choose a period value of* 15 *for interface η_4,* i.e., *we let $\eta_4 = \langle 15, 7.1445, 8.9446\rangle$. For different period values of the interface of component $\mathcal{C}_5$, we first transform interfaces η_3 and η_4 into periodic tasks using Definition 5.7. We then generate EDP interface η_5 for component $\mathcal{C}_5$. Figure 5.9 also shows the plot of η_5 for different period values. Observe that the total resource bandwidth of η_1 (period* 13*) and η_2 (period* 27*) is $\frac{3}{13} + \frac{6.95}{27} = 0.4881$, and the bandwidth of resource model η_4 for periods up to* 19 *is also* 0.4881. *This indicates that our compositional analysis technique does not incur any resource overheads for these periods. Similarly, the total resource bandwidth of η_3 (period* 20*) and η_4 (period* 15*) is $\frac{2}{20} + \frac{7.1445}{15} = 0.5763$, and the bandwidth of resource model η_5 for certain periods ($1, 5$, etc.) is the same.*



(a) Resource bandwidth Vs. Period

(b) Deadline Vs. Period

Figure 5.9: EDP interfaces η_4 and η_5

5.6 Conclusions and future work

In this chapter we proposed compositional schedulability analysis techniques based on the EDP resource model. We gave efficient algorithms to generate bandwidth optimal EDP interfaces for elementary components. Similarly, for analysis of non-elementary components, we proposed demand-supply optimal transformations from EDP interfaces to periodic tasks. In comparison to periodic resource model based interfaces, we have shown that EDP model based interfaces have smaller resource bandwidth. Also, demand-supply optimal transformations from EDP models to periodic tasks result in smaller overall resource demand for hierarchical systems, when compared to periodic model based analysis.

Techniques presented in this chapter do not support interface reuse. Development of EDP model based interfaces that are not only efficient in terms of resource utilization, but also support incremental analysis, is an important area for future work.

Chapter 6

On the complexity of generating optimal component interfaces

6.1 Introduction

Many studies have focused on the development of analysis techniques for hierarchical real-time systems. They have used resource models (*e.g.*, periodic [97, 80, 102, 54], bounded-delay [56, 101], EDP in Chapter 5) and demand bound functions [114, 107] to abstract component workloads. Various extensions to these studies have also been proposed [2, 42, 85, 43, 30, 58]. However, there is no work that characterizes the notion of *optimal resource utilization* for such systems. Informally, given a hierarchical system, optimal resource utilization refers to a quantitative measure of the minimum total amount of resource required by this system. Without knowledge of this measure, it is not possible to quantitatively assess the performance of various analysis techniques in terms of resource overhead incurred by them. Although previous studies have developed local component-level resource overhead upper bounds for interfaces [102], there is no global system-level measure for the same. Additionally, this characterization will give insight into the sources of resource overhead in existing analysis techniques, and may lead to elimination of some of those sources.

In this chapter we address the issue of defining various notions of optimal resource utilization in hierarchical systems. Specifically, we identify two notions of this optimality; *load-based optimality* also denoted as *load optimality*, and *demand-based optimality* also denoted as *demand optimality*. Intuitively, a component interface is load optimal, whenever the amount of resource required by the interface is the same as the average resource requirements of component workload. On the other hand, a component interface is demand optimal, whenever the amount of resource required by the interface is the same as the actual resource demand of component workload. In addition to defining these notions, we also develop techniques that generate such optimal interfaces. Furthermore, since hierarchical systems can be either open (components are partially specified) or closed (complete knowledge of all the components in the system), we separately define and generate optimal interfaces for each of these cases.

Assuming component workloads are comprised of constrained deadline periodic tasks, we show that load optimal interfaces, for both open and closed hierarchical systems, can be generated in time pseudo-polynomial in input size. Each interface is represented by a single constrained deadline periodic task, and hence the size of such an interface is constant in comparison to the input specification. We then show that demand optimal interfaces are hard to generate. For open systems, our interface generation technique has exponential complexity in time and size of interfaces. Using an example, we show that this complexity cannot be avoided; the interface has exponentially larger number of tasks in comparison to number of tasks in the underlying component. Likewise, for closed systems, we formulate a non-convex optimization problem with exponentially many constraints as part of our interface generation technique. The size of resulting interfaces are also exponential in comparison to the input specification.

Techniques presented in this chapter provide a baseline for resource utilization in hierarchical systems; they identify the minimum resource requirements of workloads scheduled under a given scheduling hierarchy. In addition, these techniques also reveal an interesting trade-off between

resource requirements of interfaces and their size in terms of number of tasks. We show that load optimal interfaces can be specified using a single periodic task, even though, at least in open systems, no realizable resource model based interface[1] can achieve this optimality. The proposed technique for constructing demand optimal interfaces may exponentially increase the number of tasks in the interface, compared to the number of tasks in the underlying component workload. Although, in general, this increase is unavoidable (shown using an example), demand imposed by a set of tasks in the workload may sometimes be represented by a smaller set of tasks, reducing the size of the interface. In Section 6.4.3, we characterize some of the cases when such a reduction is possible without loss of precision in demand. It is interesting to note that resource model based interfaces offer an extreme case of such reduction, essentially over-approximating resource demand and collapsing the entire workload into a single task. The optimality characterization presented here, in turn, offers the possibility of a trade-off between over-approximation of demand and interface size.

This chapter is organized as follows: In Section 6.2 we state our assumptions regarding the system model, and also define the two notions of optimal resource utilization. In Section 6.3, we develop load optimal component interfaces for both open and closed hierarchical systems. In Section 6.4 we develop demand optimal component interfaces, focusing on open systems in Section 6.4.1 and on closed systems in Section 6.4.2. We conclude this chapter and discuss future work in Section 6.5.

[1]Resource models with non-zero parameters.

6.2 Notions of optimal resource utilization

6.2.1 System model assumptions

In this chapter we assume that task sets in the workload of a component are specified using the constrained deadline periodic task model described in Section 2.1.1. Given a periodic task set $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$, we denote its utilization as $U_{\mathcal{T}} = \sum_{i=1}^{n} \frac{\mathrm{C}_i}{\mathrm{T}_i}$, and without loss of generality assume $\mathrm{D}_1 \leq \ldots \leq \mathrm{D}_n$. We consider periodic task systems as a first step towards addressing the issue of characterizing optimal resource utilization in hierarchical systems. We also assume that the hierarchical system is scheduled on a generalized uniprocessor platform having *bandwidth* $b\,(0 < b \leq 1)$. This platform is guaranteed to provide $b \times t$ units of processor capacity in every t time units.

Schedulability conditions, which are described in Theorems 2.3 and 2.4 in Section 2.1.5, can be modified to account for the generalized uniprocessor platform as follows.

Theorem 6.1 (Schedulability under EDF (uniprocessor with bandwidth b)) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of periodic tasks. Component $\mathcal{C} = \langle \mathcal{T}, \mathrm{EDF} \rangle$ is schedulable on an uniprocessor platform having bandwidth b iff*

$$\forall t \text{ s.t. } 0 < t \leq L, \mathrm{dbf}_{\mathcal{C}}(t) \leq b \times t, \tag{6.1}$$

where $L = \min\left\{\mathrm{LCM} + \max_{i=1,\ldots,n} \mathrm{D}_i, \frac{U_{\mathcal{T}}(\max_{i=1,\ldots,n}(\mathrm{T}_i - \mathrm{D}_i))}{b - U_{\mathcal{T}}}\right\}$, LCM being the least common multiple of $\mathrm{T}_1, \ldots, \mathrm{T}_n$.

If $b \geq \mathrm{LOAD}_{\mathcal{C}} = \max_{t \in (0,L]} \frac{\mathrm{dbf}_{\mathcal{C}}(t)}{t}$, then the uniprocessor platform can successfully schedule component $\mathcal{C}$. Following theorem gives the schedulability condition for an elementary component $\mathcal{C}$, when it is scheduled under DM on an uniprocessor platform having bandwidth b.

Theorem 6.2 (Schedulability under DM (uniprocessor with bandwidth b)) *Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of periodic tasks. Component $\mathcal{C} = \langle \mathcal{T}, \mathrm{DM} \rangle$ is schedulable on an*

uniprocessor platform having bandwidth b iff

$$\forall i, \exists t \in [0, \mathrm{D}_i] \ \ s.t. \ \ \mathrm{rbf}_{\mathcal{C},i}(t) \leq b \times t \tag{6.2}$$

If $b \geq \mathrm{LOAD}_{\mathcal{C}} = \max_{i=1,\ldots,n} \min_{t \in (0,\mathrm{D}_i]} \frac{\mathrm{rbf}_{(\mathcal{C},i)}(t)}{t}$, then the uniprocessor platform can successfully schedule component $\mathcal{C}$. These conditions are agnostic to preemptions incurred by task set $\mathcal{T}$, and hence we ignore preemption overhead in this chapter.

Recall the definition of a component interface given in Section 2.1.5. We reproduce this definition below for easy access.

Definition 6.1 (Component interface) *We define the interface of a task set to be the task set itself. Consider a component $\mathcal{C} = \langle \mathcal{W}, \mathcal{S} \rangle$ with $\mathcal{W} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$. Let $\mathcal{C}$ itself be scheduled under $\mathcal{S}'$, and $\mathcal{I}_{\mathcal{W}}$ denote the set of interfaces of workload $\mathcal{W}$. $\mathcal{I}_{\mathcal{C}}$ is an interface for $\mathcal{C}$ iff $\mathcal{I}_{\mathcal{C}}$ is schedulable under $\mathcal{S}'$ implies $\mathcal{I}_{\mathcal{W}}$ is schedulable under $\mathcal{S}$, where schedulability is determined using Theorems 2.3 and 2.4. In this definition, we assume that $\mathcal{I}_{\mathcal{W}}$ executes under $\mathcal{S}$ whenever $\mathcal{I}_{\mathcal{C}}$ is scheduled by $\mathcal{S}'$.*

Thus, given a component $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \mathcal{S} \rangle$, the fundamental question in scheduling $\mathcal{C}$ is, "What is the interface that each $\mathcal{C}_i$ must present to $\mathcal{S}$?".

Synchronization. An important aspect of hierarchical systems, both open and closed, is that of synchronization between components. Given a component $\mathcal{C} = \langle \mathcal{W}, \mathcal{S} \rangle$, release of jobs in the underlying workload of some component $\mathcal{C}_i \in \mathcal{W}$, need not be synchronized with the release of jobs in the underlying workload of any other component in $\mathcal{W}$ (*Horizontal Synchronization*). Note that horizontal synchronization can be enforced in closed systems, but this is not the case in open systems. This means that our interface generation technique cannot make any assumptions regarding synchronization between interfaces of components in $\mathcal{W}$. Now, suppose we synchronize

the release time of first job in a component in $\mathcal{W}$ with the release time of first job in its interface (*Vertical Synchronization*). Then, observe that this forced vertical synchronization does not enforce any horizontal synchronization constraints in the system. In other words, we can vertically synchronize every component in $\mathcal{W}$ with its interface, without assuming any (horizontal) synchronization between components in $\mathcal{W}$ or between interfaces of those components. Therefore, we assume vertical synchronization in the interface generation technique presented in this chapter. Note that this assumption only synchronizes the release times of first jobs in component and interface, and does not enforce any synchronization between other jobs.

6.2.2 Optimal resource utilization

The *feasibility load* $\text{LOAD}_{\mathcal{I}_\mathcal{C}}$ of a component interface $\mathcal{I}_\mathcal{C}$, is the smallest bandwidth required from an uniprocessor platform to successfully schedule tasks in $\mathcal{I}_\mathcal{C}$ under some scheduler. Similarly, given a set of interfaces $\mathcal{I} = \{\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}\}$ and a scheduler $\mathcal{S}$, the *schedulability load* $\text{LOAD}_{\mathcal{I},\mathcal{S}}$ is the smallest bandwidth required from an uniprocessor platform to successfully schedule $\mathcal{I}$ under $\mathcal{S}$. For instance, if $\mathcal{I}$ comprises of constrained deadline periodic tasks and $\mathcal{S}$ is EDF or DM, then we have presented these loads in the previous section. Also, a non-elementary component $\mathcal{C} = \langle\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \mathcal{S}\rangle$ is said to be *feasible* on an uniprocessor platform having bandwidth b, if there exists interface set $\mathcal{I} = \{\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}\}$ such that $\mathcal{I}$ is schedulable under $\mathcal{S}$ on this resource.

Load-based optimality. The first notion of optimal resource utilization we define is called *load* optimality, which characterizes average resource requirements of components.

Definition 6.2 (Local load optimality) *Given a constrained deadline periodic task set, its interface (task set itself) is defined to be locally load optimal. Consider component* $\mathcal{C}_i = \langle\{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}, \mathcal{S}_i\rangle$. *If* $\mathcal{I}_i$ *denotes the set of interfaces of workload* $\{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}$, *then* $\mathcal{I}_{\mathcal{C}_i}$ *is* locally load optimal *iff* $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_i}} = \text{LOAD}_{\mathcal{I}_i,\mathcal{S}_i}$.

If interface $\mathcal{I}_{\mathcal{C}_i}$ is locally load optimal, then the interface generation process does not introduce

any overheads with respect to the average resource requirements of component $\mathcal{C}_i$. In other words, if $\mathcal{C}_i$ is feasible on an uniprocessor platform having bandwidth b, then $\mathcal{I}_{\mathcal{C}_i}$ is also feasible on that resource. There could be many possible locally load optimal interfaces for $\mathcal{C}_i$, and not all of them may result in a load optimal interface for $\mathcal{C}_i$'s parent. Therefore, for closed systems, since we have knowledge of other components scheduled with $\mathcal{C}_i$, it is useful to identify those locally load optimal interfaces of $\mathcal{C}_i$ that also result in load optimality for its parent.

Definition 6.3 (Global load optimality) *Given a constrained deadline periodic task set, its interface (task set itself) is defined to be globally load optimal. Consider component* $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}, \mathcal{S} \rangle$. *Let* $\mathcal{I} = \{\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_n}\}$ *denote the set of locally load optimal interfaces of the workload of* $\mathcal{C}$, *and* $\mathcal{I}_{\mathcal{C}}$ *denote a locally load optimal interface generated from* $\mathcal{I}$. *Also, let* $\mathcal{I}'$ *denote any other set of locally load optimal interfaces of the workload of* $\mathcal{C}$, *and* $\mathcal{I}'_{\mathcal{C}}$ *denote the locally load optimal interface generated from* $\mathcal{I}'$. *Then, each interface* $\mathcal{I}_{\mathcal{C}_i} \in \mathcal{I}$ *is* globally load optimal *iff* $\text{LOAD}_{\mathcal{I}_{\mathcal{C}}} \leq \text{LOAD}_{\mathcal{I}'_{\mathcal{C}}}$ *for every* $\mathcal{I}'_{\mathcal{C}}$.

In Section 6.3 we present a technique that generates globally load optimal interfaces for both open and closed systems. Thus, we are able to generate globally load optimal interfaces for $\mathcal{C}_i$, even without knowledge of other components scheduled with it. Following theorem describes the significance of load optimal interfaces.

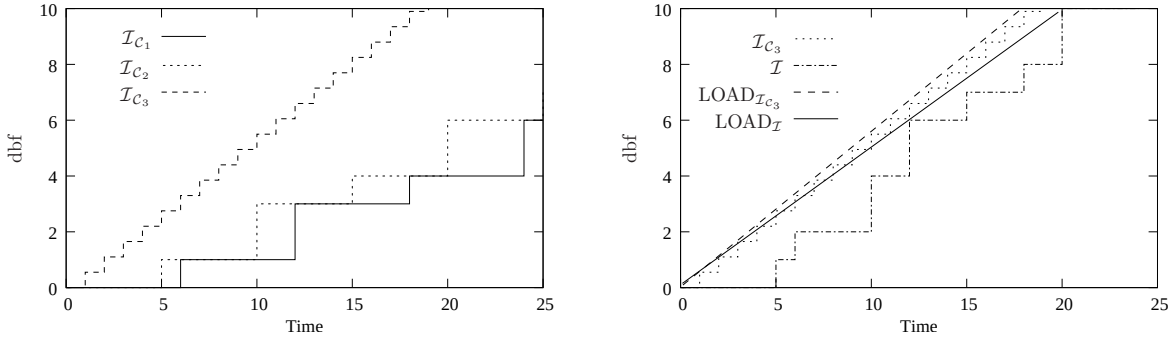
Theorem 6.3 *Consider a hierarchical system* $\mathcal{H} = \langle \mathcal{C}, \mathcal{S} \rangle$, *with* $\mathcal{C}_1, \ldots, \mathcal{C}_m$ *denoting all the components in the tree rooted at* $\mathcal{C}$. *Let interfaces* $\mathcal{I} = \{\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_m}\}$ *of all these components be globally load optimal. Also, let* $\mathcal{I}_{\mathcal{C}}$ *denote a load optimal interface for* $\mathcal{C}$ *generated from* $\mathcal{I}$. *If each* $\mathcal{C}_i$ *is scheduled exclusively on an uniprocessor having bandwidth* $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_i}}$ $(= b_i)$, *then* $\mathcal{C}$ *is not schedulable on any uniprocessor having bandwidth* b *that is smaller than* $\text{LOAD}_{\mathcal{I}_{\mathcal{C}}}$.

Proof We prove this theorem by induction on the height of node $\mathcal{C}$.

Base Case [$\mathcal{C}$ is a task set] : In this case $\text{LOAD}_{\mathcal{I}_\mathcal{C}} = \text{LOAD}_\mathcal{C}$, and by definition $\text{LOAD}_\mathcal{C}$ is the smallest bandwidth required from an uniprocessor platform to schedule $\mathcal{C}$.

Induction Case [$\mathcal{C}$ is at height $n+1$] : Let $\mathcal{S}'$ denote the scheduler used by component $\mathcal{C}$ to schedule its workload, and $\mathcal{I}_n$ denote the set of globally load optimal interfaces of all components at height n. Then, from Definitions 6.2 and 6.3 we get $\text{LOAD}_{\mathcal{I}_\mathcal{C}} = \text{LOAD}_{\mathcal{I}_n,\mathcal{S}'}$ and $\text{LOAD}_{\mathcal{I}_\mathcal{C}} \leq \text{LOAD}_{\mathcal{I}'_n,\mathcal{S}'}$, where $\mathcal{I}'_n$ denotes any other set of locally load optimal interfaces of all components at height n. Also, by definition, $\text{LOAD}_{\mathcal{I}_\mathcal{C}}$ denotes the smallest bandwidth required from an uniprocessor platform to successfully schedule component $\langle \mathcal{I}_n, \mathcal{S}' \rangle$. Consider an uniprocessor platform having bandwidth $b < \text{LOAD}_{\mathcal{I}_\mathcal{C}}$. Then, some interface $\mathcal{I}_{\mathcal{C}_l} \in \mathcal{I}_n$ is not schedulable on this platform, *i.e.*, bandwidth of the uniprocessor available for scheduling workloads in $\mathcal{I}_{\mathcal{C}_l}$ is smaller than $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_l}}$. But from induction hypothesis, $b_l = \text{LOAD}_{\mathcal{I}_{\mathcal{C}_l}}$ denotes the smallest bandwidth required from an uniprocessor platform to successfully schedule component $\mathcal{C}_l$. This proves the result. □

Demand-based optimality. Consider a hierarchical system defined as follows. $\mathcal{C}_1 = \langle \{(6,1,6),(12,1,12)\}, \text{EDF} \rangle$, $\mathcal{C}_2 = \langle \{(5,1,5),(10,1,10)\}, \text{EDF} \rangle$, and $\mathcal{C}_3 = \langle \{\mathcal{C}_1, \mathcal{C}_2\}, \text{EDF} \rangle$. Consider the interfaces $\mathcal{I}_{\mathcal{C}_1} = (1, 0.25, 1), \mathcal{I}_{\mathcal{C}_2} = (1, 0.3, 1)$, and $\mathcal{I}_{\mathcal{C}_3} = (1, 0.55, 1)$. It is easy to see that $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_1}} = \text{LOAD}_{\mathcal{C}_1}$, $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_2}} = \text{LOAD}_{\mathcal{C}_2}$, and $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_3}} = \text{LOAD}_{\mathcal{I}',\text{EDF}}$, where $\mathcal{I}' = \{\mathcal{I}_{\mathcal{C}_1}, \mathcal{I}_{\mathcal{C}_2}\}$. Hence $\mathcal{I}_{\mathcal{C}_1}$ and $\mathcal{I}_{\mathcal{C}_2}$ are globally load optimal. The demand bound functions (dbf) of these interfaces and components are plotted in Figure 6.1(a). However, as seen in Figure 6.1(b), $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_3}} > \text{LOAD}_\mathcal{I}$, where $\mathcal{I} = \{(6,1,6),(12,1,12),(5,1,5),(10,1,10)\}$. Assuming vertical synchronization, it is easy to see that total resource requirements of $\mathcal{I}$ is equal to the total resource requirements of components $\mathcal{C}_1$ and $\mathcal{C}_2$ combined. Therefore, $\mathcal{I}$ is an interface for component $\mathcal{C}_3$. This shows that even though $\mathcal{I}_{\mathcal{C}_3}$ is only feasible on an uniprocessor platform having bandwidth $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_3}}$, component $\mathcal{C}_3$ itself is schedulable on a platform having bandwidth strictly



(a) Load optimal interface

(b) Sub-optimality

Figure 6.1: Load vs. demand optimality

smaller than $\text{LOAD}_{\mathcal{I}_{\mathcal{C}_3}}$. Thus, although a load optimal interface minimizes the average resource utilization, it still incurs overheads with respect to the actual demand of underlying component.

We now introduce the notion of demand-based optimality, which characterizes interfaces that are both necessary and sufficient for component schedulability. We define two notions of demand optimality; local optimality for open systems and global optimality for closed systems. When the hierarchical system under consideration is an open system, a component in the system is not aware of other components scheduled with it. Therefore, when generating an interface for such a component, we must consider the worst-case interference from other components scheduled with it. This worst-case interference can be defined as follows.

Definition 6.4 (Zero slack assumption) *Consider component* $\mathcal{C}_i = \langle \{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}, \mathcal{S}_i \rangle$. *Let* $\mathcal{I}_i$ *denote the set of interfaces of workload* $\{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}$. *If* $\mathcal{C}_i$ *is part of an open hierarchical system and* $\mathcal{I}_i$ *is schedulable under* $\mathcal{S}_i$, *then we assume that the schedule of* $\mathcal{I}_i$ *has* zero slack. *In other words, the amount of resource supplied to* $\mathcal{C}_i$ *is such that each job in* $\mathcal{I}_i$ *finishes as late as possible, subject to satisfaction of all job deadlines.*

The following definition then gives our notion of demand optimality for open systems.

Definition 6.5 (Local demand optimality (open systems)) *Given a constrained deadline periodic task set, its interface (task set itself) is defined to be locally demand optimal. Consider*

component $\mathcal{C}_i = \langle \{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}, \mathcal{S}_i \rangle$. *Let* $\mathcal{I}_i$ *denote the set of interfaces of workload* $\{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}$. *Interface* $\mathcal{I}_{\mathcal{C}_i}$ *is locally demand optimal iff assuming zero slack for* $\mathcal{C}_i$ *(and hence for* $\mathcal{I}_{\mathcal{C}_i}$*), schedulability of* $\mathcal{I}_i$ *under* $\mathcal{S}_i$ *implies feasibility of* $\mathcal{I}_{\mathcal{C}_i}$.

If $\mathcal{I}_{\mathcal{C}_i}$ is locally demand optimal, then the interface generation process does not introduce any overheads with respect to the resource requirements of $\mathcal{I}_i$, assuming zero slack in the schedule. In Section 6.4.1 we generate locally demand optimal interfaces for components in open systems. Locally demand optimal interfaces need not always lead to global optimality, because the actual interference from other components may be smaller than the worst-case scenario considered here. Therefore, in closed systems, where we have accurate knowledge of this interference, we are able to generate interfaces that achieve global demand optimality defined as follows.

Definition 6.6 (Global demand optimality (closed systems)) *Given a constrained deadline periodic task set, its interface (task set itself) is defined to be globally demand optimal. Consider a hierarchical system* $\langle \mathcal{C}, \mathcal{S} \rangle$. *Let* $\mathcal{I}_{\mathcal{C}}$ *denote an interface for* $\mathcal{C}$ *generated using some set of interfaces for all components in* $\mathcal{C}$. $\mathcal{I}_{\mathcal{C}}$ *is globally demand optimal if and only if, whenever there exists interfaces for all the components in* $\mathcal{C}$ *such that the components are schedulable using those interfaces,* $\mathcal{I}_{\mathcal{C}}$ *is feasible.*

Intuitively, the significance of globally demand optimal interfaces can be explained as follows. Given a hierarchical system, if it is possible to schedule all the tasks in the system using some set of interfaces, then globally demand optimal interfaces will also schedule them. Note that in Definition 6.1 we use schedulability of $\mathcal{I}_{\mathcal{C}_i}$ under $\mathcal{S}$ to identify whether $\mathcal{I}_{\mathcal{C}_i}$ is an interface for $\mathcal{C}_i$. On the other hand, in the optimality definitions above we only use feasibility of interface $\mathcal{I}_{\mathcal{C}_i}$. This is reasonable, because feasibility of component $\mathcal{C}_i$ need not necessarily imply schedulability of $\mathcal{C}_i$ under $\mathcal{S}$.

6.3 Load optimal interfaces

6.3.1 Interface generation technique

The following definition presents an interface that is globally load optimal in both open and closed hierarchical systems.

Definition 6.7 (Schedulability load based abstraction) *If $\mathcal{C}_i$ is a constrained deadline periodic task set then abstraction $\mathcal{I}_{\mathcal{C}_i} = \mathcal{C}_i$. Otherwise $\mathcal{I}_{\mathcal{C}_i} = \{\tau_i = (1, \text{LOAD}_{\mathcal{W}_{\mathcal{C}_i}, \mathcal{S}_i}, 1)\}$, where $\mathcal{S}_i$ denotes scheduler used by $\mathcal{C}_i$, and $\mathcal{W}_{\mathcal{C}_i}$ denotes the set of schedulability load based abstractions of $\mathcal{C}_i$'s children. τ_i is a periodic task, and the release time of its first job coincides with the release time of the first job in component $\mathcal{C}_i$ (vertical synchronization).*

We now prove that the schedulability load based abstraction $\mathcal{I}_{\mathcal{C}_i}$ is a globally load optimal interface for component $\mathcal{C}_i$.

Lemma 6.4 *Given a component $\mathcal{C}_i = \langle \mathcal{T}, \mathcal{S}_i \rangle$, where $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is a constrained deadline periodic task set, $\mathcal{I}_{\mathcal{C}_i}$ as given by Definition 6.7 is an interface for $\mathcal{C}_i$.*

Proof Let us define $L_{\mathcal{T}} = \{t\}_i$ to be the sequence of non-decreasing time units measured from the point when the first job in $\mathcal{T}$ is released. Each t_i in the sequence is such that there are no jobs of $\mathcal{T}$ with non-zero remaining executions, assuming $\mathcal{T}$ executes under $\mathcal{S}_i$ whenever $\mathcal{I}_{\mathcal{C}_i}$ executes. We prove this lemma by induction on this sequence.

Base case $[t_1]$ **:** In the interval $(0, t_1]$, jobs of $\mathcal{T}$ are always executing whenever $\mathcal{I}_{\mathcal{C}_i}$ is executing. This means that the amount of resource used by $\mathcal{T}$ in $(0, t_1]$ is exactly equal to the amount of resource used by $\mathcal{I}_{\mathcal{C}_i}$ in that interval. By Definition 6.7, the amount of resource used by $\mathcal{I}_{\mathcal{C}_i}$ in an interval $(0, t']$, for all $t' \leq t_1$, is $\text{LOAD}_{\mathcal{T}, \mathcal{S}_i} \times t'$ (vertical synchronization).

If $\mathcal{S}_i = \text{EDF}$ the amount of resource required by $\mathcal{T}$ to meet all job deadlines up to t' is upper bounded by $\text{dbf}_{\mathcal{C}_i}(t')$. The result follows because $\text{LOAD}_{\mathcal{T}, \mathcal{S}_i} \times t' \geq \text{dbf}_{\mathcal{C}_i}(t')$ by definition. If

$\mathcal{S}_i = \text{DM}$, the amount of resource required by $\mathcal{T}$ to meet all job deadlines up to t', such that these are jobs of task τ_k, is upper bounded by $t' \times \min_{t_k \in (0, \mathrm{D}_k]} \frac{\mathrm{rbf}_{\mathcal{C}_i,k}(t_k)}{t_k}$. The result then follows because $\mathrm{LOAD}_{\mathcal{T},\mathcal{S}_i} \times t' \geq t' \times \max_{k=1,\ldots,n} \min_{t_k \in (0,\mathrm{D}_k]} \frac{\mathrm{rbf}_{\mathcal{C}_i,k}(t_k)}{t_k}$.

Induction case $[t_{n+1}]$ **:** By induction hypothesis, all job deadlines in the interval $(0, t_n]$ are satisfied by jobs of $\mathcal{I}_{\mathcal{C}_i}$ released before t_n. Observe that $t_{n+1} \geq t_n + 1$, and the first release of any job of $\mathcal{T}$ after t_n (t_r) can only occur at or after $t_n + 1$. Also, since period of task τ_i in interface $\mathcal{I}_{\mathcal{C}_i}$ is 1, some release of job of $\mathcal{I}_{\mathcal{C}_i}$ after t_n will coincide with t_r. Using logic similar to the base case, we can show that all deadlines of jobs of $\mathcal{T}$ in the interval $(t_r, t_{n+1}]$, are satisfied by jobs of $\mathcal{I}_{\mathcal{C}_i}$ released in that interval. This combined with the induction hypothesis proves the result. □

Theorem 6.5 *Interface $\mathcal{I}_{\mathcal{C}_i}$ generated using Definition 6.7 is an interface for component $\mathcal{C}_i$.*

Proof We prove this theorem by induction on the height of $\mathcal{C}_i$ in the underlying sub-tree rooted at $\mathcal{C}_i$.

Base case $[\mathcal{C}_i$ **is a task set]** **:** Direct from Definitions 6.1 and 6.7.

Induction case $[\mathcal{C}_i$ **is at height** $n+1]$ **:** If $n = 1$, then $\mathcal{C}_i$'s child is a task set, and from induction hypothesis and Lemma 6.4 we directly get the result. If $n > 1$, then let $\mathcal{C}_i = \langle \{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}, \mathcal{S}_i \rangle$. Lemma 6.4 can again be used to show that $\mathcal{I}_{\mathcal{C}_i}$ is an interface for the component $\langle \mathcal{I}_i, \mathcal{S}_i \rangle$, where $\mathcal{I}_i$ is the set of interfaces of workload $\{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_m}\}$. But, by induction hypothesis, each $\mathcal{I}_{\mathcal{C}_{i_j}} \in \mathcal{I}_i$ is itself an interface for $\mathcal{C}_{i_j}$, and therefore $\mathcal{I}_{\mathcal{C}_i}$ is an interface for $\mathcal{C}_i$. □

Finally, we now prove that interface $\mathcal{I}_{\mathcal{C}_i}$ is globally load optimal.

Theorem 6.6 *Given component $\mathcal{C} = \langle \{\mathcal{C}_1, \ldots \mathcal{C}_i, \ldots, \mathcal{C}_n\}, \mathcal{S} \rangle$, if interfaces $\mathcal{I}_{\mathcal{C}}$ and $\mathcal{I} = \{\mathcal{I}_{\mathcal{C}_1}, \ldots, \mathcal{I}_{\mathcal{C}_i}, \ldots, \mathcal{I}_{\mathcal{C}_n}\}$ are as given by Definition 6.7, then $\mathcal{I}_{\mathcal{C}_i}$ is a globally load optimal interface.*

Proof We prove this theorem by induction on the height of $\mathcal{C}_i$ in the sub-tree rooted at $\mathcal{C}$.

Base Case [$\mathcal{C}_i$ is a task set] : In this case, $\mathcal{I}_{\mathcal{C}_i}$ is globally load optimal directly from Definitions 6.3 and 6.7.

Induction Case [$\mathcal{C}_i$ is at height $n+1$] : From induction hypothesis we get that each $\mathcal{I}_{\mathcal{C}_i}$ is locally load optimal (interfaces of $\mathcal{C}_i$'s children of are globally load optimal). Since $\mathcal{W}_{\mathcal{C}} = \mathcal{I}$, $\text{LOAD}_{\mathcal{W}_{\mathcal{C}},\mathcal{S}} = \text{LOAD}_{\mathcal{I},\mathcal{S}}$. Also, since $\mathcal{I}_{\mathcal{C}} = (1, \text{LOAD}_{\mathcal{W}_{\mathcal{C}},\mathcal{S}}, 1)$, $\text{LOAD}_{\mathcal{I}_{\mathcal{C}}} = \text{LOAD}_{\mathcal{I},\mathcal{S}}$ and $\mathcal{I}_{\mathcal{C}}$ is locally load optimal from Definition 6.2. Furthermore, $\text{LOAD}_{\mathcal{I},\mathcal{S}} = \text{LOAD}_{\mathcal{I}}$, the feasibility load of $\mathcal{I}$. This is because all the tasks in $\mathcal{I}$ have period and deadline equal to 1, and EDF and DM are optimal schedulers for such task sets. Therefore, $\text{LOAD}_{\mathcal{I}_{\mathcal{C}}} = \text{LOAD}_{\mathcal{I}} \leq \text{LOAD}_{\mathcal{I}',\mathcal{S}}$, where $\mathcal{I}'$ denotes any other set of locally load optimal interfaces for $\mathcal{C}$'s workload. □

6.3.2 Discussions

Complexity. Interfaces given in Definition 6.7 can be computed in pseudo-polynomial time with respect to input specification. $\text{LOAD}_{\mathcal{W}_{\mathcal{C}_i},\mathcal{S}_i}$ can be computed using Equation (6.1) or (6.2). Since these equations must be evaluated for all values of t in the range $(0, L]$ under EDF, and $(0, \text{D}_j]$ for each task $\tau_j \in \mathcal{W}_{\mathcal{C}_i}$ under DM, interface $\mathcal{I}_{\mathcal{C}_i}$ can be generated in pseudo-polynomial time. Also, interface $\mathcal{I}_{\mathcal{C}_i}$ only has $\mathcal{O}(1)$ storage requirements when compared to the input specification.

Task model. Although we have considered periodic tasks in this chapter, interface generation technique in Definition 6.7 also generates load optimal interfaces for constrained deadline sporadic tasks. The only modifications required in Definition 6.7 are that, (1) task τ_i is sporadic, and (2) τ_i is released whenever there are unfinished jobs active in $\mathcal{C}_i$, subject to these releases satisfying the minimum separation criteria. Using similar proof techniques, we can show that Theorems 6.5 and 6.6 hold for such interfaces as well.

Comparison to resource model based interfaces. In an open system, interface parameters of one component is not known to other components in the system. For this case, previous studies have shown that the feasibility load of interface generated using bounded delay [56, 101],

periodic [80, 102], or EDP (Chapter 5) resource models, is equal to the schedulability load of underlying component only when period $\Pi = 0$ for periodic/EDP models and delay $\delta = 0$ for bounded delay models (see Corollaries 6.1 and 6.2 in [102] and Theorems 4 and 5 in [101]). Note Π or $\delta = 0$ indicates that the interface is not realizable, because EDF and DM cannot schedule tasks generated from such models. Hence, these resource model based interfaces are not load optimal. The reason for this sub-optimality is lack of vertical synchronization between the component and its interface.

On the other hand, in a closed system, interface parameters of one component are known to other components in the system. Then, assuming interfaces use EDP resource models, it is possible to choose the same period value for all the interfaces in the system. In addition to this period restriction, if we also enforce the condition $\Delta = \Theta$ in each interface, then the resulting interfaces will be load optimal. Correctness of this statement follows from the fact that (1) in any time interval of length Π these interfaces guarantee Θ units of resource, and (2) transformation from EDP models to periodic tasks is demand-supply optimal (see Equation (5.1) and Definition 5.4 and in Chapter 5).

6.4 Demand optimal interfaces

Before we present techniques that generate demand optimal interfaces, we introduce some additional definitions. A constrained deadline, asynchronous periodic task set is specified as $\mathcal{T} = \{\tau_1 = (\mathrm{O}_1, \mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{O}_n, \mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$, where each τ_i is a periodic task with offset O_i, exact separation T_i, worst case execution requirement C_i, and relative deadline D_i, such that $\mathrm{C}_i \leq \mathrm{D}_i \leq \mathrm{T}_i$. For each task τ_i, its jobs are released at times $\mathrm{O}_i, \mathrm{O}_i + \mathrm{T}_i, \mathrm{O}_i + 2\,\mathrm{T}_i, \ldots$, and each job requires C_i units of resource within D_i time units. Without loss of generality we assume $\mathrm{D}_1 \leq \ldots \leq \mathrm{D}_n$.

Techniques presented in this section may produce interfaces that are comprised of constrained deadline, asynchronous periodic tasks, even when underlying task sets in the hierarchical system are synchronous. Therefore, for keeping the presentation simple, we also assume that task sets in the hierarchical system are comprised of such asynchronous tasks. Furthermore, in all the interfaces we generate, each task τ_i satisfies the property $\mathrm{O}_i + \mathrm{D}_i \leq \mathrm{T}_i$. Hence, we assume this property for all the asynchronous tasks that we consider.

6.4.1 Interfaces for open systems

We now present a technique to generate a locally demand optimal interface for a component $\mathcal{C}_i$, assuming only the knowledge of interfaces of its children (open system).

Definition 6.8 (Absolute demand based abstraction (open system)) *If $\mathcal{C}_i$ is a task set then abstraction $\mathcal{I}_{\mathcal{C}_i} = \mathcal{C}_i$. Otherwise let $\mathcal{S}_i$ denote the scheduler used by $\mathcal{C}_i$, $\mathcal{W}_{\mathcal{C}_i} = \{\tau_1 = (\mathrm{O}_1, \mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_m = (\mathrm{O}_m, \mathrm{T}_m, \mathrm{C}_m, \mathrm{D}_m)\}$ denote the set of absolute demand based abstractions of $\mathcal{C}_i$'s children, and $\mathcal{S}$ denote the scheduler under which $\mathcal{C}_i$ is scheduled.*

1. ***$\mathcal{S}$=DM, $\mathcal{S}_i$=DM:*** $\mathcal{I}_{\mathcal{C}_i} = \mathcal{W}_{\mathcal{C}_i}$.

2. ***$\mathcal{S}$=DM, $\mathcal{S}_i$=EDF:*** $\mathcal{I}_{\mathcal{C}_i} = \mathcal{W}_{\mathcal{C}_i}$.

3. ***$\mathcal{S}$=EDF, $\mathcal{S}_i$=EDF:*** $\mathcal{I}_{\mathcal{C}_i} = \mathcal{W}_{\mathcal{C}_i}$.

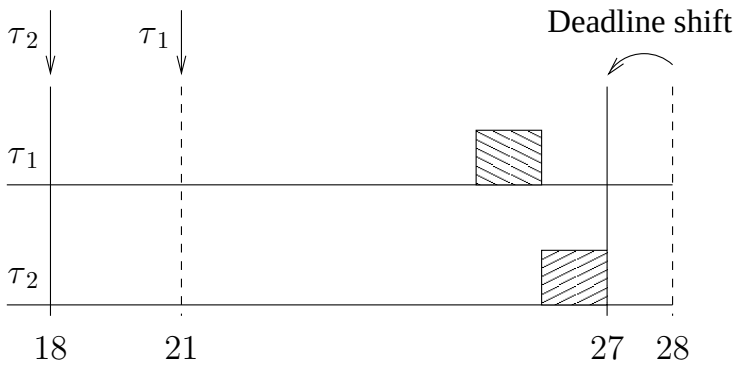
4. ***$\mathcal{S}$=EDF, $\mathcal{S}_i$=DM:*** $\mathcal{I}_{\mathcal{C}_i} = \{\tau_{t_k} = (t_k, LCM, \mathrm{C}_k, \mathrm{D}_{t_k}) | \, \mathrm{O}_k \leq t_k \leq LCM; \mathrm{T}_k$ *divides* $(t_k - \mathrm{O}_k); k = 1, \ldots, m\}$, *LCM being the least common multiple of* $\mathrm{T}_1, \ldots, \mathrm{T}_m$. $\mathrm{D}_{t_k} = \min\{F_{t_k}, t_k + \mathrm{D}_k, \min_{j>k}\{t_j + \mathrm{D}_{t_j}\}\} - t_k$, *where t_j is the smallest time satisfying $t_k < t_j + \mathrm{D}_{t_j}$ for each j, and F_{t_k} is the latest release time of some job of $\tau_1, \ldots, \tau_{k-1}$ satisfying the condition*

$$r < t_k + \mathrm{D}_k \;\; \textit{and} \; t_k + \mathrm{D}_k - r > \sum_{j<k} \left[\left\lceil \frac{t_k + \mathrm{D}_k - O_j}{\mathrm{T}_j} \right\rceil - \left\lceil \frac{r - \mathrm{O}_j}{\mathrm{T}_j} \right\rceil \right] \mathrm{C}_j$$

Release time of the first job in interface $\mathcal{I}_{\mathcal{C}_i}$ coincides with the release time of the first job in component $\mathcal{C}_i$ (vertical synchronization).

t_1	D_{t_1}	t_2	D_{t_2}
0	7	0	9
7	2	9	9
14	4	18	9
21	6	27	8
28	7	36	9
35	7	45	9
42	3	54	9
49	5		
56	7		

(a) Deadlines of τ_{t_1}, τ_{t_2}



(b) Partial schedule of component $\mathcal{C}_1$

Figure 6.2: Deadline generation in example system

Since each deadline D_{t_k} in the above definition depends on deadlines of lower priority jobs, we generate them in reverse order of task priorities, *i.e.*, we first generate deadlines of tasks τ_{t_m}, then $\tau_{t_{m-1}}$, and so on. Also, F_{t_k} denotes the latest possible finish time of τ_k's job released at t_k. As per zero slack assumption, F_{t_k} is the earliest time when this job of τ_k can finish its executions.

Example 6.1 *We illustrate this interface generation process for a simple example. Consider components* $\mathcal{C}_1 = \langle\{\tau_1 = (0,7,1,7), \tau_2 = (0,9,1,9)\}, \mathcal{S}_1\rangle$, *and* $\mathcal{C} = \langle\{\mathcal{C}_1, \mathcal{C}_2\}, \mathcal{S}\rangle$. *Here,* $\mathcal{C}_2$ *is unknown (open system), and therefore we assume that it can interfere with the execution of* $\mathcal{C}_1$ *in an adversarial manner (zero slack assumption). If* $\mathcal{S}$ = DM *and* $\mathcal{S}_1$ = EDF/DM, *or* $\mathcal{S}$ = EDF *and* $\mathcal{S}_1$ = EDF, *then from Definition 6.8 we get that* $\mathcal{I}_{\mathcal{C}_1} = \{\tau_1, \tau_2\}$. *On the other hand, if* $\mathcal{S}$ = EDF *and* $\mathcal{S}_1$ = DM, *then* $\mathcal{I}_{\mathcal{C}_1} = \{\tau_{t_1} = (t_1, 63, 1, \mathrm{D}_{t_1}), \tau_{t_2} = (t_2, 63, 1, \mathrm{D}_{t_2}) | t_1 = 0, 7, 14, 21, 28, 35, 42, 49, 56; t_2 = 0, 9, 18, 27, 36, 45, 54\}$. *Values of* D_{t_1} *and* D_{t_2} *are listed in Table 6.2(a). For instance, consider the job of* τ_2 *released at time* 18, *with deadline at* 27. *Since schedule of* $\mathcal{C}_1$ *has zero slack, we assume that this job finishes its execution requirements only by time* 27 *(its latest possible finish time). Then, job of* τ_1 *released at time* 21 *must also finish its execution by time* 27 *under* DM. *Therefore, as shown in Figure 6.2(b), we decrease the job's deadline before presenting it to the higher level* EDF *scheduler.*

Following theorem proves that interfaces generated by Definition 6.8 are locally demand optimal.

Theorem 6.7 *Consider component* $\mathcal{C}_i = \langle \{\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_n}\}, \mathcal{S}_i \rangle$ *scheduled under scheduler* $\mathcal{S}$. *Let* $\mathcal{T} = \{\tau_1 = (\mathrm{O}_1, \mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_m = (\mathrm{O}_m, \mathrm{T}_m, \mathrm{C}_m, \mathrm{D}_m)\}$ *denote the set of tasks in interfaces* $\mathcal{I}_{\mathcal{C}_{i_1}}, \ldots, \mathcal{I}_{\mathcal{C}_{i_n}}$. $\mathcal{I}_{\mathcal{C}_i}$ *given in Definition 6.8 is a locally demand optimal interface for* $\mathcal{C}_i$.

Proof We prove this result for the following cases depending on schedulers $\mathcal{S}_i$ and $\mathcal{S}$.

Case 1 ($\mathcal{S}$=DM and $\mathcal{S}_i$=DM) : By Definition 6.8, $\mathcal{I}_{\mathcal{C}_i} = \mathcal{T}$. In this case, relative priority ordering of tasks in $\mathcal{I}_{\mathcal{C}_i}$ under $\mathcal{S}$ and $\mathcal{T}$ under $\mathcal{S}_i$ are the same. Therefore, because of vertical synchronization, whenever a task $\tau_i \in \mathcal{I}_{\mathcal{C}_i}$ is scheduled under $\mathcal{S}$, task $\tau_i \in \mathcal{T}$ is also scheduled under $\mathcal{S}_i$. Hence, $\mathcal{I}_{\mathcal{C}_i}$ is a locally demand optimal interface for $\mathcal{C}_i$.

Case 2 ($\mathcal{S}$=DM and $\mathcal{S}_i$=EDF) : In this case, by Definition 6.8, $\mathcal{I}_{\mathcal{C}_i} = \mathcal{T}$. Also, if $\mathcal{T}$ is schedulable under DM, then it is also schedulable under EDF [78]. Therefore, because of vertical synchronization, $\mathcal{I}_{\mathcal{C}_i}$ is an interface for $\mathcal{C}_i$. Local demand optimality follows because schedulability of $\mathcal{T}$ under $\mathcal{S}_i$ implies feasibility of $\mathcal{I}_{\mathcal{C}_i}$, even under zero slack assumption.

Case 3 ($\mathcal{S}$=EDF and $\mathcal{S}_i$=EDF) : Similar to Case 1 above.

Case 4 ($\mathcal{S}$=EDF and $\mathcal{S}_i$=DM) : First we prove that $\mathcal{I}_{\mathcal{C}_i}$ is an interface. When $\mathcal{I}_{\mathcal{C}_i}$ is scheduled under $\mathcal{S}$, if a job of task τ_{t_k} has the same deadline as a job of task τ_{t_l} with $k < l$, then we assume that the tie is broken in favor of τ_{t_k}'s job. Then, from Definition 6.8 we can see that whenever a job J of task $\tau_{t_k} \in \mathcal{I}_{\mathcal{C}_i}$ executes under $\mathcal{S}$, a job J' of task $\tau_k \in \mathcal{T}$ having the same release time as that of J executes under $\mathcal{S}_i$. Furthermore, since $\mathrm{D}_{t_k} \leq \mathrm{D}_k$, deadline of J is smaller than or equal to deadline of J'. This and vertical synchronization, together prove that $\mathcal{I}_{\mathcal{C}_i}$ is an interface for component $\mathcal{C}_i$.

We now prove the local demand optimality of $\mathcal{I}_{\mathcal{C}_i}$. Let job J (mentioned above) miss its deadline at $t_k + \mathrm{D}_{t_k}$. Then, in the interval $(t_k - 1, t_k]$, some job of task $\tau_j \in \mathcal{T}$ with $j \leq k$ was executing under scheduler $\mathcal{S}_i$. Now, from Definition 6.8 we know that either (1) there exists some job of some task $\tau_l \in \mathcal{T}$ with $l \geq k$, such that the deadline of this job is $t_k + \mathrm{D}_{t_k}$, or (2)

$t_k + \mathrm{D}_{t_k} = F_{t_l}$ for some t_l and $l \geq k$. Let J' denote such a job so that it has the lowest priority among all jobs that satisfy (1) or (2). Now, since we assume zero slack, J' cannot finish before $t_k + \mathrm{D}_{t_k}$, and hence it will miss its deadline under $\mathcal{S}_i$. This proves that $\mathcal{I}_{\mathcal{C}_i}$ is locally demand optimal.

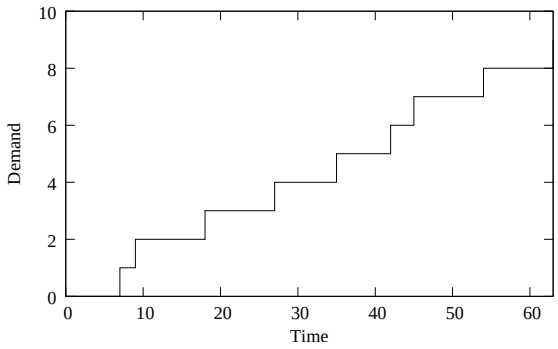
In this proof, we assumed that if a job of task τ_{t_k} has the same deadline as a job of task τ_{t_l} with $k < l$, then the tie is broken in favor of τ_{t_k}'s job under $\mathcal{S}$. However, it is easy to see that similar arguments can be used to prove the result when this assumption does not hold. □

Consider the system discussed in Example 6.1. As shown by Theorem 6.7, for each value of t_1, the value of D_{1,t_1} listed in Table 6.2(a) is both necessary and sufficient to guarantee schedulability of $\mathcal{C}_1$. Each $\mathrm{D}_{1,t_1} \leq \mathrm{D}_1$, and hence it is sufficient. Also, since the interference of component $\mathcal{C}_2$ on $\mathcal{C}_1$ is unknown, it is always possible for jobs of τ_2 to finish only by their deadlines. Then, to guarantee schedulability of the job of τ_2 released at t_2 (with deadline $t_2 + \mathrm{D}_2$), it is necessary that any job of τ_1 released before $t_2 + \mathrm{D}_2$ should also finish by $t_2 + \mathrm{D}_2$. Hence, the values of D_{1,t_1} listed in Table 6.2(a) are also necessary for schedulability of $\mathcal{C}_1$.

This example also shows that, in general, size of a locally demand optimal interface can be exponentially larger than the input specification. This can be explained as follows. Consider task τ_1 of component $\mathcal{C}_1$. The amount of resource required by this task in $(0, 63]$ is given by Figure 6.3(b) (this demand function assumes zero slack). Also, release constraints on these demands are given in Table 6.3(a). As shown by Theorem 6.7, these demands and release constraints are exact in the sense that they are necessary and sufficient to guarantee schedulability of τ_1. Then, any locally demand optimal interface must reproduce this demand function and release constraints exactly to abstract τ_1. Suppose a periodic task $\tau = (\mathrm{O}, \mathrm{T}, \mathrm{C}, \mathrm{D})$ is used to abstract the resource requirements of some jobs of τ_1. Then, O and D must be such that $(\mathrm{O}, \mathrm{O} + \mathrm{D}]$ is one of the entries in Table 6.3(a). Also, T must be such that, for all k, $\mathrm{O} + k\,\mathrm{T} = l\,\mathrm{LCM} + a$

Interval	Demand
$(0, 7]$	1
$(7, 9]$	1
$(14, 18]$	1
$(21, 27]$	1
$(28, 35]$	1
$(35, 42]$	1
$(42, 45]$	1
$(49, 54]$	1
$(56, 63]$	1

(a) Demand intervals of task τ_1



(b) Demand function of task τ_1

Figure 6.3: Demand of task τ_1 in component $\mathcal{C}_1$

and $\mathrm{O} + k\,\mathrm{T} + \mathrm{D} = l\,\mathrm{LCM} + b$ for some $l \geq 0$ and entry $(a, b]$ in the table. It is easy to see that these properties do not hold for any $\mathrm{T} < 63$. This means that task τ can be used to abstract the demand of only one job of τ_1 in the interval $(0, 63]$. Therefore, at least $\frac{\mathrm{LCM}}{\mathrm{T}_1} = 9$ tasks are required to abstract the demand of all jobs of τ_1. This shows that the exponential complexity of demand optimal interfaces cannot be avoided for the example under consideration.

6.4.2 Interfaces for closed systems

Definition 6.8 generates a locally demand optimal component interface by assuming zero slack. Although it is the best we can hope for in open systems, this is not the case for closed systems. Since we have complete knowledge of the hierarchical system, we should be able to generate interfaces that are globally demand optimal as in Definition 6.6. Below we present one such technique.

Definition 6.9 (Absolute demand based abstraction (closed system)) *If $\mathcal{C}_i$ is a task set then abstraction $\mathcal{I}_{\mathcal{C}_i} = \mathcal{C}_i$. Otherwise let $\mathcal{S}_i$ denote the scheduler used by $\mathcal{C}_i$, $\mathcal{W}_{\mathcal{C}_i} = \{\tau_1 = (\mathrm{O}_1, \mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_m = (\mathrm{O}_m, \mathrm{T}_m, \mathrm{C}_m, \mathrm{D}_m)\}$ denote the set of absolute demand based abstractions of $\mathcal{C}_i$'s children, and $\mathcal{S}$ denote the scheduler under which $\mathcal{C}_i$ is scheduled.*

1. ***$\mathcal{S}$=DM, $\mathcal{S}_i$=DM:*** *$\mathcal{I}_{\mathcal{C}_i} = \mathcal{W}_{\mathcal{C}_i}$.*

2. $\mathcal{S}$=**DM,** $\mathcal{S}_i$=**EDF:** $\mathcal{I}_{\mathcal{C}_i} = \mathcal{W}_{\mathcal{C}_i}$.

3. $\mathcal{S}$=**EDF,** $\mathcal{S}_i$=**EDF:** $\mathcal{I}_{\mathcal{C}_i} = \mathcal{W}_{\mathcal{C}_i}$.

4. $\mathcal{S}$=**EDF,** $\mathcal{S}_i$=**DM:** $\mathcal{I}_{\mathcal{C}_i} = \{\tau_{t_k} = (t_k, LCM, \mathrm{C}_k, \mathrm{D}_{t_k}) | \, \mathrm{O}_k \leq t_k \leq \mathrm{LCM}; \mathrm{T}_k$ *divides* $(t_k - \mathrm{O}_k); k = 1, \ldots, m\}$, LCM *being the least common multiple of* $\mathrm{T}_1, \ldots, \mathrm{T}_m$.

Release time of the first job in interface $\mathcal{I}_{\mathcal{C}_i}$ *coincides with the release time of the first job in component* $\mathcal{C}_i$ *(vertical synchronization). Also, deadlines* D_{t_k} *are given by the following optimization problem.*

Objective: *Minimize* $\mathrm{LOAD}_{\mathcal{I}_{\mathcal{C}_i}}$

Subject to: *For each pair of components* $\mathcal{C}_a$ *and* $\mathcal{C}_b$ *such that scheduler of* $\mathcal{C}_a$ *is* DM, $\mathcal{C}_a$ *is a child of* $\mathcal{C}_b$, *and scheduler of* $\mathcal{C}_b$ *is* EDF, *and for each unknown deadline* D_{t_k} *in* $\mathcal{I}_{\mathcal{C}_a}$,

- $0 \leq \mathrm{D}_{t_k} \leq \mathrm{D}_k$
- $|(t_k + \mathrm{D}_{t_k} - (t_j + \mathrm{D}_{t_j})) + (t_k - (t_j + \mathrm{D}_{t_j}))| \geq \mathrm{D}_{t_k}$, *for each* $j > k$ *and* $\tau_{t_j} \in \mathcal{I}_{\mathcal{C}_a}$

Consider the second constraint in the optimization problem described above. Since $j > k$, jobs of τ_j have a lower priority than those of τ_k under DM. Therefore, it is required that no deadline of τ_{t_j} lie in the interval $[t_k, t_k + \mathrm{D}_{t_k})$. Otherwise, under EDF, jobs of τ_{t_j} can have higher priority than those of τ_{t_k}. It is easy to see that the second constraint enforces this condition. Suppose some deadline of τ_{t_j} lies in the interval $[t_k, t_k + \mathrm{D}_{t_k})$. Then, $(t_k + \mathrm{D}_{t_k} - (t_j + \mathrm{D}_{t_j}))$ is positive but smaller than D_{t_k}, and $(t_k - (t_j + \mathrm{D}_{t_j})) \leq 0$. In this case, the second constraint can never be satisfied.

Theorem 6.8 *Consider a hierarchical real-time system* $\langle \mathcal{C}_i, \mathcal{S} \rangle$. *Abstraction* $\mathcal{I}_{\mathcal{C}_i}$ *as given in Definition 6.9 is a globally demand optimal interface for* $\mathcal{C}_i$.

Proof Using similar arguments as those in Theorem 6.7, we can show that for each component $\mathcal{C}$ in the hierarchical system, $\mathcal{I}_{\mathcal{C}}$ generated in Definition 6.9 is an interface.

Consider any pair of components $\mathcal{C}_a$ and $\mathcal{C}_b$ such that $\mathcal{C}_a$ is a child of $\mathcal{C}_b$. If their scheduling policies correspond to Cases (1),(2) or (3) in Definition 6.9, then $\mathcal{C}_a$'s interface is identical to its workload. In this case, the resource demand of $\mathcal{I}_{\mathcal{C}_a}$ is identical to that of $\mathcal{C}_a$'s workload, and therefore the resource demand of any other interface must be at least as much as that of $\mathcal{I}_{\mathcal{C}_a}$. On the other hand, if their scheduling policies correspond to Case (4), then we introduce a new deadline variable D_{t_k} for each job J in $\mathcal{C}_a$'s workload released in $(0, \mathrm{LCM}]$. Furthermore, we add constraints on D_{t_k} in the optimization problem. These constraints allow D_{t_k} to take any value between 0 and D_k, except that deadlines of jobs released before D_{t_k} and having priority higher than J, are forced to be at most D_{t_k}. Since $\mathcal{C}_a$'s scheduler in this case is DM and $\mathcal{C}_b$'s scheduler is EDF, these constraints are necessary (otherwise the resulting interface will not satisfy Definition 2.10). This also means that the resource demand of any other interface for $\mathcal{C}_a$ must be at least as much as the resource demand generated by these constraints. The result then follows, because $\mathrm{LOAD}_{\mathcal{I}_{\mathcal{C}_i}}$ is the smallest among all interfaces that can be generated for $\mathcal{C}_i$ satisfying all these demand constraints. □

Interface generated by Definition 6.9 can also have size proportional to LCM of task periods in workload interfaces. Since interference from other components can be adversarial even in closed systems (zero slack assumption), example in Section 6.4.1 illustrates necessity of increased interface size for this case as well.

6.4.3 Discussions

Complexity. Technique in Definition 6.8 has a constant running time and storage requirement for all four cases, provided only symbolic representation of interfaces are used. However, for a scheduler to be able to schedule an interface, we must enumerate all the tasks in the interface.

Then, cases (1), (2), and (3) have a running time which is linear in input size, and case (4) has an exponential running time complexity. This is because LCM of task periods in $\mathcal{W}_{\mathcal{C}_i}$ can be exponentially larger than the task parameters. Also, the storage requirements are linear in cases (1), (2) and (3), and exponential in case (4).

For closed systems, cases (1), (2), and (3) in Definition 6.9, are identical to the corresponding cases in open systems. However, when there exists a pair of components that satisfy case (4), complexity of computing interfaces is equal to the complexity of solving the optimization problem in Definition 6.9. This problem is non-convex[2] and has exponentially many deadline variables, and hence hard to solve exactly [110]. Also, similar to the open systems case, sizes of resulting interfaces can be exponentially larger than input specification.

Task Model. Consider a hierarchical system $\langle \mathcal{C}_i, \mathcal{S} \rangle$ such that tasks in the underlying subtree rooted at $\mathcal{C}_i$ are constrained deadline sporadic tasks. Definitions 6.8 and 6.9 will generate a demand optimal interface for $\mathcal{C}_i$, provided all parent–child component pairs in $\mathcal{C}_i$ belong to cases (1), (2), or (3). However, even if a single pair satisfies case (4), then our techniques will not generate a demand optimal interface for $\mathcal{C}_i$. This is because, in these definitions, when we compute deadline D_{t_k} for task τ_{t_k}, we assume accurate knowledge of the deadlines of all lower priority jobs. This assumption does not hold for sporadic task systems. It is an open problem to generate demand optimal interfaces when workloads in hierarchical systems are comprised of sporadic tasks.

Comparison between locally demand optimal and load optimal interfaces. Consider a component $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$, where $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ is a constrained deadline periodic task set. From Lemma 6.4 we get that $(1, \mathrm{LOAD}_{\mathcal{C}}, 1)$ is a load optimal interface for $\mathcal{C}$. Additionally, if the first job of all the tasks in $\mathcal{T}$ are released synchronously, then we can also show that $\mathcal{I}_{\mathcal{C}} = (\mathrm{GCD}, \mathrm{LOAD}_{\mathcal{C}} * \mathrm{GCD}, \mathrm{GCD})$ is a load optimal

[2]LOAD depends on dbf and rbf which have floor/ceiling functions.

interface for $\mathcal{C}$, where GCD is the greatest common divisor of $\mathrm{T}_1, \ldots, \mathrm{T}_n, \mathrm{D}_1, \ldots, \mathrm{D}_n$. Consider the case GCD $= \mathrm{T}_1 = \ldots = \mathrm{T}_n = \mathrm{D}_1 = \ldots = \mathrm{D}_n$. In this case, it is easy to see that $\mathcal{I}_\mathcal{C}$ also satisfies the notion of local demand optimality (Definition 6.5). This is because $\mathrm{dbf}_\mathcal{C} = \mathrm{dbf}_{\mathcal{I}_\mathcal{C}}$. On the other hand, if $\mathrm{T}_i \neq \mathrm{D}_i$ for some i or $\mathrm{T}_i \neq \mathrm{T}_j$ for some i and j, then we can show there exists t such that $\mathrm{dbf}_\mathcal{C}(t) < \mathrm{LOAD}_\mathcal{C} \times t = \mathrm{dbf}_{\mathcal{I}_\mathcal{C}}$ and $t = k \times \mathrm{GCD}$ for some integer k. For instance, if $\mathrm{T}_i \neq \mathrm{D}_i$ for some i, then at $t = \mathrm{LCM}$ this property holds, where LCM denotes least common multiple of $\mathrm{T}_1, \ldots, \mathrm{T}_n$. Similarly, if $\mathrm{T}_i \neq \mathrm{T}_j$ for some i and j and $\mathrm{D}_i = \mathrm{T}_i$ for all i, then at $t = \min_{i=1,\ldots,n} \mathrm{T}_i$ this property holds. Hence, $\mathcal{I}_\mathcal{C}$ does not satisfy the property of local demand optimality whenever $\mathrm{T}_i \neq \mathrm{D}_i$ for some i or $\mathrm{T}_i \neq \mathrm{T}_j$ for some i and j. Similar arguments also apply to components that use DM scheduler. Thus, load optimality also results in local demand optimality in one extremely restrictive case.

Comparison between globally demand optimal and load optimal interfaces. Consider a component $\mathcal{C}$, with $\mathcal{C}_1, \ldots, \mathcal{C}_m$ denoting all the elementary components in the tree rooted at $\mathcal{C}$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be the only components in $\mathcal{C}$ with periodic tasks in their workloads, and $\mathcal{S}_1, \ldots, \mathcal{S}_m$ denote their respective schedulers such that each $\mathcal{S}_i = \mathrm{EDF}$. Also, let $\mathcal{I}_\mathcal{C}$ denote a load optimal interface for component $\mathcal{C}$. Then, assuming all interfaces in this system have period one, we get that $\mathrm{LOAD}_{\mathcal{I}_\mathcal{C}} = \sum_{i=1}^{m} \mathrm{LOAD}_{\mathcal{C}_i,\mathcal{S}_i}$ from Theorem 6.3. Now, suppose there exists a time t such that for each i, $\mathrm{LOAD}_{\mathcal{C}_i,\mathcal{S}_i} \times t = \mathrm{dbf}_{\mathcal{C}_i}(t)$. Note that all the task sets have their maximum load at the same time instant. Then, in this case, $\mathcal{I}_\mathcal{C}$ is also globally demand optimal, because $\sum_{i=1}^{m} \mathrm{LOAD}_{\mathcal{C}_i,\mathcal{S}_i}$ is indeed the minimum bandwidth required from an uniprocessor platform to schedule $\mathcal{C}$. On the other hand, if such a t does not exist or if some $\mathcal{S}_i$ is DM, then $\sum_{i=1}^{m} \mathrm{LOAD}_{\mathcal{C}_i,\mathcal{S}_i}$ can be strictly larger than the minimum required bandwidth (for instance, see example in Section 6.2.2). In this case, $\mathcal{I}_\mathcal{C}$ is not globally demand optimal.

6.5 Conclusions and future work

In this chapter we introduced two notions of optimal resource utilization in hierarchical systems. We proposed efficient techniques to generate load optimal interfaces, which characterize optimality with respect to average resource requirements. Each load optimal interface comprises of a single task, and hence has $\mathcal{O}(1)$ storage requirements when compared to the input specification. We also proposed techniques to generate demand optimal interfaces (optimality with respect to absolute resource demand), for both open and closed systems. As long as there does not exist a parent–child component pair such that the child uses DM and parent uses EDF, these techniques are efficient in time and space (linear in running time and storage requirements). On the other hand, even if a single such pair exists, then interfaces are hard to generate using our techniques (exponential in time and space). We showed using an example that this complexity cannot be avoided.

We focused on constrained deadline periodic task systems in this chapter. Although the load optimal interfaces that we generate are also load optimal for constrained deadline sporadic task systems, this is not the case for demand optimal interfaces. It is an open problem to generate demand optimal interfaces for such sporadic task systems.

Although, in general, the complexity of a demand optimal interface is unavoidable (as shown by the example in Section 6.4.1), demand imposed by a set of tasks in the interface may sometimes be represented by a smaller set of tasks, reducing the size of the interface. An interesting area of future work is to characterize the cases when such a reduction is possible.

Chapter 7

Case study: ARINC-653, avionics real-time OS specification

7.1 Introduction

ARINC standards, developed and adopted by the *Engineering Standards for Avionics and Cabin Systems* committee, deliver substantial benefits to airlines and aviation industry by promoting competition, providing inter changeability, and reducing life-cycle costs for avionics and cabin systems. In particular, the 600 series ARINC specifications and reports define enabling technologies that provide a design foundation for digital avionics systems. Within the 600 series, this chapter deals with ARINC specification 653-2, part I [55] (henceforth referred to as ARINC-653), which defines a general-purpose Application/Executive (APEX) software interface between the operating system of an avionics computer and the application software. These interface requirements are defined in a manner that enables application software to control its scheduling, communication, and status of internal processing elements.

As described in ARINC-653 standards, the embedded system of an aircraft comprises of one or more *core modules* connected with each other using switched Ethernet. A partial model of this

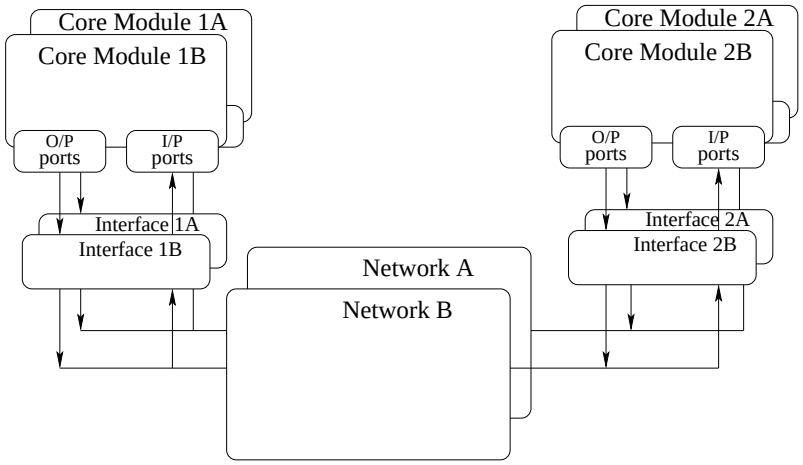


Figure 7.1: Embedded system of an aircraft

distributed system is shown in Figure 7.1, where the elements are duplicated once. As shown in the figure, core modules communicate using i/p and o/p ports which are connected to the network through interfaces. In an aircraft one can expect multiple such core modules to be connected over a switched network, supporting arbitrary redundancy. Each core module is a hardware platform that consists of one or more processors among other things. Core modules support independent execution of one or more avionics applications. They provide space and temporal partitioning for applications, which enables fault containment and simplifies verification. Each independently executing function in an application is called a *partition*, and each partition in turn is comprised of one or more *processes* representing its real-time resource demand. As shown in Figure 7.2, the core module software architecture comprises of: (1) application layer software including partitions and processes within partitions, (2) APEX (application executive) interface, and (3) an OS kernel, among other things. The APEX interface defines a set of API that applications can use to control their scheduling, communication, or status information. This interface isolates the application layer from hardware platform of core modules, and hence supports easy migration. The OS kernel is responsible for providing functionality specified by the APEX interface.

The workload on a single processor in a core module can be described as a two-level hierarchical real-time system. Each partition comprises of one or more processes that are scheduled among

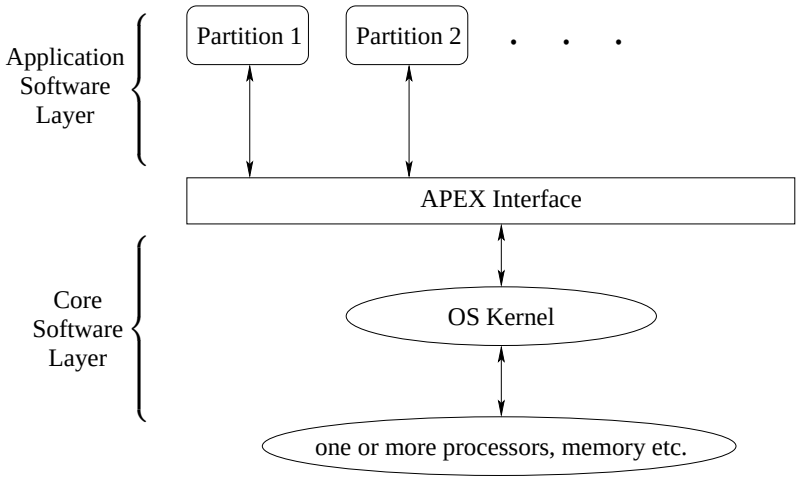


Figure 7.2: Core module software architecture

themselves using a local partition specific scheduler. All the partitions that are allocated to the same processor are then scheduled among themselves using a global partition level scheduler. For example, Figure 7.3 shows one such system, where partitions $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are scheduled under DM on an uniprocessor platform. Each partition $\mathcal{P}_i$ in turn is comprised of processes $\tau_{i,1}, \ldots, \tau_{i,m_i}$, also scheduled under DM. Processes are periodic tasks and can communicate with each other using input and output ports either asynchronously or synchronously (Figure 7.1 illustrates these communication ports). Sequences of such communicating processes form dependency chains, and designers can specify end-to-end latency bounds for them. Processes within a partition can block each other using *semaphores* giving rise to blocking overhead. Furthermore, processes and partitions can also be preempted by higher priority processes and partitions respectively, resulting in preemption related overheads.

There are several problems related to the hierarchical system described above that must be addressed. For scheduling partitions, it is desirable to abstract the communication dependencies between processes using parameters like offsets, jitter, and constrained deadlines. This simplifies a global processor and network scheduling problem into several local single processor scheduling problems. The process deadlines must also guarantee satisfaction of end-to-end latency bounds

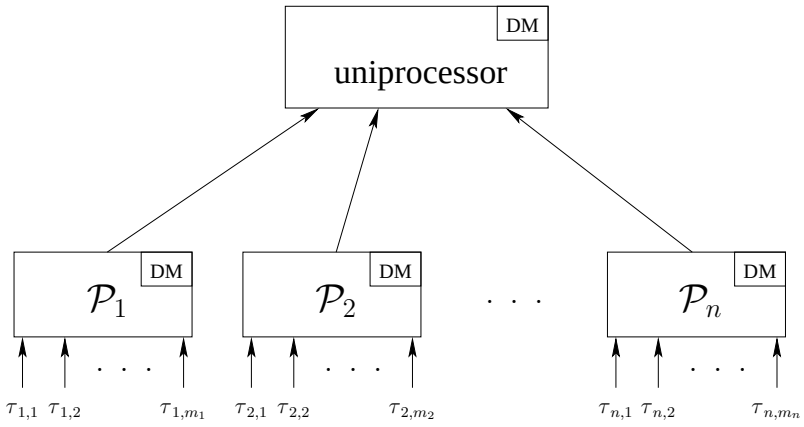


Figure 7.3: Scheduling hierarchy for partitions

specified by the designer. Given such processes we must then generate scheduling parameters for partitions. The resulting partition schedule must provide sufficient processor capacity to schedule processes within partitions. Furthermore, these scheduling parameters must also account for blocking and preemption overheads incurred by processes and partitions.

Among other things, each partition is typically developed by third party vendors, and hence our analysis techniques must support abstraction of partition workloads. This allows vendors to expose only so much information about partitions as is required for the analysis, thus protecting their intellectual property. Observing that each partition is essentially an elementary component in a hierarchical system, one could abstract its workload using resource model based component interfaces described in Chapters 2, 4, and 5. However, before we can use these analysis techniques, we must modify them to handle partition specific issues like communication dependencies, and blocking and preemption overheads. In this chapter, we assume that communication dependencies and end-to-end latency bounds are abstracted using existing techniques into process parameters like offset, jitter, and constrained deadline (see [108, 92, 103, 61, 98, 34, 69, 1, 60, 111]). Note that although this chapter does not present solutions to this problem, it is however important, because it motivates the inclusion of aforementioned process parameters.

Contributions. In this chapter we model ARINC-653 as a two-level hierarchical system,

and develop compositional analysis techniques for the same. This is a principled approach for scheduling ARINC-653 partitions that provides separation of concerns among different application vendors, and therefore should facilitate system integration. In particular, our contributions can be summarized as follows:

- We extend and improve existing periodic and EDP resource model based analysis techniques to take into account (a) process communications modeled as offsets, jitter, and constrained deadlines, and (b) process preemption and blocking overheads. Section 7.3 presents this solution, and illustrates its effectiveness using real-world workloads.

- We develop techniques to schedule partitions using their interfaces, taking into account preemption overheads incurred by partitions. Specifically, in Section 7.4, we present a technique to count the exact number of preemptions incurred by partitions in the partition level schedule. Mataix *et. al.* [36] compute the number of preemptions when partitions are scheduled under a fixed priority scheduler. However, unlike our technique which counts the preemptions exactly, they only present an upper bound.

In the remaining parts of this chapter, we introduce notations for our system model in Section 7.2, and conclude the chapter in Section 7.5.

Related work. Traditionally the partition scheduling problem has been addressed in an ad-hoc fashion based on interactions between the system designer and vendors who provide the applications. Although many different ARINC-653 platforms exist (see [65, 115]), there is little work on automatic scheduling of partitions [72, 74, 91]. Kinnan *et. al.* [72] only provide preliminary heuristic guidance, and the other studies [74, 91] use constraint-based approaches to look at combined network and processor scheduling. In contrast to this high complexity holistic analysis, we present an efficient compositional analysis technique that also protects intellectual property of partition vendors.

7.2 System model

Partitions and processes. Each partition has an associated period that identifies the frequency with which it executes, *i.e.*, it represents the interface period. Typically, this period is derived from the periods of processes that form the partition. In this work, we assume that partitions are scheduled among themselves using DM scheduler. This enables us to generate a static partition level schedule at design time, as required by the specification. Processes within a partition are assumed to be periodic tasks[1]. ARINC-653 allows processes to be scheduled using preemptive, fixed priority schedulers, and hence we assume that each partition also uses DM to schedule processes in its workload.

As discussed in the introduction, we assume that communication dependencies and end-to-end latency requirements are modeled with process offsets, jitter, and constrained deadlines. Hence, each process can be specified as a constrained deadline periodic task $\tau = (\mathrm{O}, \mathrm{J}, \mathrm{T}, \mathrm{C}, \mathrm{D})$, where T is period, C is worst case execution time, D($\leq$ T) is deadline, O is offset, and J is jitter. A job of this process is *dispatched* at time instant $x\,\mathrm{T} + \mathrm{O}$ for some non-negative integer x, and it will be *released* for execution at any time in the interval $[x\,\mathrm{T} + \mathrm{O}, x\,\mathrm{T} + \mathrm{O} + \mathrm{J}]$. For such a process it is reasonable to assume that $\mathrm{O} \leq \mathrm{D}$ [108]. Furthermore, we denote as $\langle\{\tau_1, \ldots, \tau_n\}, \mathrm{DM}\rangle$, a partition $\mathcal{P}$ comprising of processes $\tau_1, \ldots, \tau_n$ and using scheduler DM. Without loss of generality we assume that τ_i has higher priority than τ_j for all $i < j$.

In addition to the restrictions specified so far, we make the following assumptions for the system described herein. These assumptions have been verified to exist in avionics systems.

- The processes within a partition, and hence the partition itself, cannot be distributed over multiple processors.

- Periods of partitions that are scheduled on the same processor are *harmonic*. A set of

[1]Partitions with aperiodic processes also exist in avionics systems, but they are scheduled as background workload. Hence, we ignore such processes in this work.

numbers $\{T_1, \ldots, T_n\}$ is harmonic if and only if, for all i and j, either T_i divides T_j or T_j divides T_i. Note that this assumption does not prevent processes from having non-harmonic periods.

- Processes in a partition cannot block processes in another partition. This is because mutual exclusion based on semaphores require use of shared memory which can only happen within a partition.

7.3 Partition interface generation

In this section we propose techniques to compute a periodic or EDP resource model based interface for partition $\mathcal{P}$. We assume that $\Pi_{\mathcal{P}}$ denotes the interface period specified by system designer for $\mathcal{P}$. We first briefly discuss shortcomings of existing techniques presented in Chapters 2 and 5, and then develop techniques that overcome these shortcomings.

7.3.1 Inadequacy of existing analysis

Recall that compositional analysis techniques presented in Chapters 2 and 5 consider processes with zero offset and jitter values[2]. The request bound function (rbf) for a partition whose workload is comprised of such processes is given by Definition 2.9 in Section 2.1.5. When processes have non-zero jitter, Tindell and Clark have derived a critical arrival pattern which can be used to compute rbf [109]. For a given process, the critical arrival pattern corresponds to an arrival sequence that generates the maximum demand from all its higher priority processes. In the arrival pattern proposed by Tindell and Clark, each higher priority process is released simultaneously with the process under consideration, incurring maximum possible jitter. All future instances of these higher priority processes are released as soon as possible, *i.e.*, they incur zero jitter. Furthermore, the process under consideration itself is assumed to incur maximum possible jitter. Figure 7.4

[2]Periodic or sporadic tasks in elementary components.

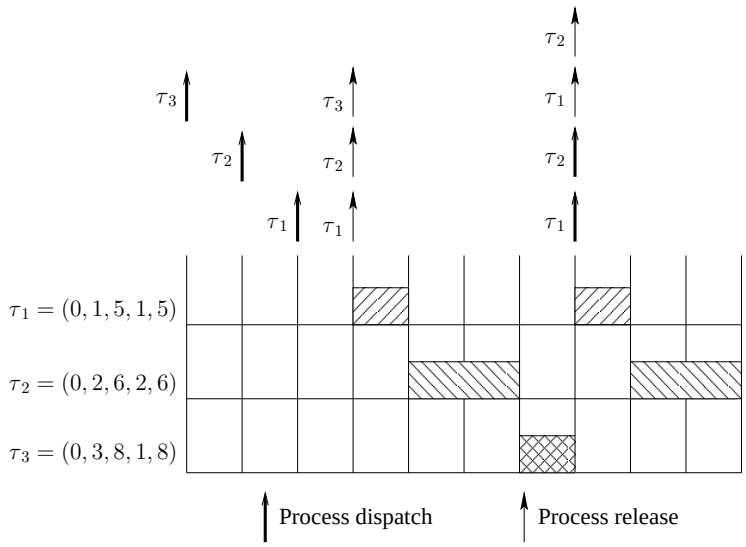


Figure 7.4: Critical arrival pattern for processes with non-zero jitter

illustrates this critical arrival pattern for process τ_3 in an example partition $\langle\{\tau_1, \tau_2, \tau_3\}, \text{DM}\rangle$. Thus, for a process τ_i with zero offset but non-zero jitter, its request bound function $\text{rbf}_{\mathcal{P},i}$ can be specified as

$$\text{rbf}_{\mathcal{P},i}(t) = \sum_{j=1}^{i} \left(\left\lceil \frac{t + \text{J}_j}{\text{T}_j} \right\rceil \text{C}_j \right) \tag{7.1}$$

When processes in a partition have zero offset and jitter values, conditions for schedulability of the partition using a periodic or EDP resource model are presented in Sections 2.2.2 and 5.3 respectively. These conditions can be easily extended to consider processes with non-zero jitter and we present it in the following theorem.

Theorem 7.1 (Schedulability under DM (processes with jitter)) *Let* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *denote a set of processes, where for each* i, $\tau_i = (0, \text{J}_i, \text{T}_i, \text{C}_i, \text{D}_i)$. *Partition* $\mathcal{P} = \langle \mathcal{T}, \text{DM} \rangle$ *is schedulable using a periodic or EDP resource model* R *iff*

$$\forall i : 1 \leq i \leq n, \exists t_i \in [0, \text{D}_i - \text{J}_i] \text{ s.t. } \text{rbf}_{\mathcal{P},i}(t_i) \leq \text{sbf}_R(t_i),$$

where $\text{rbf}_{\mathcal{P},i}$ *is given by Equation* (7.1).

Proof Direct from Theorems 2.6 and 5.2, and properties of critical arrival pattern presented by Tindell and Clark [109]. □

Periodic or EDP resource model based interface for partition $\mathcal{P}$ can be generated using Theorem 7.1. For this purpose, we assume that the period of resource model R is equal to $\Pi_{\mathcal{P}}$. If R is a periodic resource model, then techniques presented in Section 2.2.3 can be used to develop the interface, and if R is an EDP model, then techniques presented in Section 5.4 can be used for this purpose. However, as described in the introduction, processes can be more accurately modeled using non-zero offset and jitter values. Then, a major drawback in using the aforementioned technique is that Theorem 7.1 only gives sufficient schedulability conditions. This follows from the fact that the critical arrival pattern used by Equation (7.1) is pessimistic for processes with non-zero offset values. Additionally, the aforementioned technique does not take into account preemption and blocking overheads incurred by processes.

In the following sections we extend Theorem 7.1 to accommodate processes with non-zero offsets, as well as to account for blocking and preemption overheads. Recollect from Section 7.2 that all the partitions scheduled on a processor are assumed to have harmonic interface periods. This observation leads to a tighter supply bound function for periodic resource models when compared to the general case. Therefore, we first present a new sbf for periodic resource models, and then extend Theorem 7.1.

7.3.2 sbf under harmonic interface periods

Recall from the discussion in Section 2.2.3 that a periodic interface $\phi = \langle \Pi, \Theta \rangle$ is transformed into a periodic task $\tau_\phi = (\Pi, \Theta, \Pi)$, before it is presented to the parent scheduler. Note that the period of model ϕ and task τ_ϕ are identical. For the ARINC-653 partitions, this means that partitions scheduled on a processor are abstracted into periodic tasks with harmonic periods. When such implicit deadline periodic tasks are scheduled under DM, every job of a task is scheduled in the

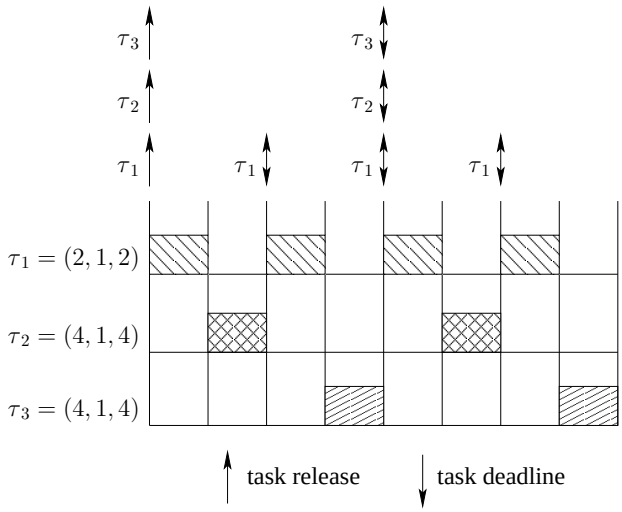


Figure 7.5: Schedule of tasks with harmonic periods and implicit deadlines

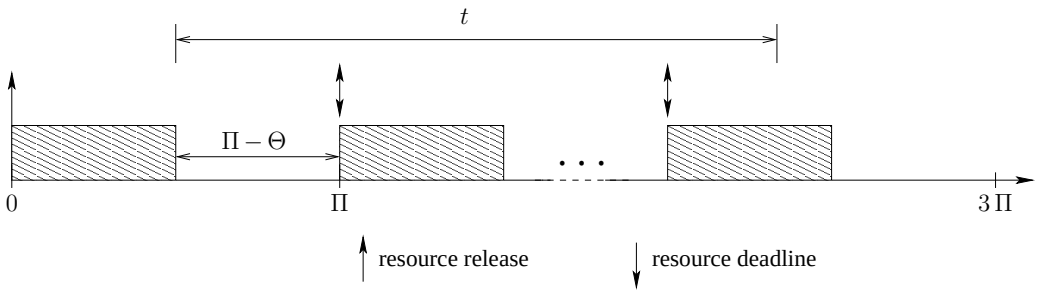


Figure 7.6: Resource schedule corresponding to improved value of $\text{sbf}_\phi(t)$

same time instants within its execution window. This follows from the observation that whenever a job of a task is released, all the higher priority tasks also release a job at the same time instant. For example, Figure 7.5 shows the schedule for a periodic task set $\{\tau_1 = (2, 1, 2), \tau_2 = (4, 1, 4), \tau_3 = (4, 1, 4)\}$. Every job of task τ_3 is scheduled in an identical manner within its execution window.

Whenever task τ_ϕ is executing, the resource is available for use by periodic model ϕ. This means that resource supply allocations for ϕ also occur in an identical manner within intervals $(n\,\Pi, (n+1)\,\Pi]$, for all $n \geq 0$. In other words, the blackout interval in sbf can never exceed $\Pi - \Theta$. For the example shown in Figure 7.5, assuming task τ_3 is transformed from a periodic resource model $\phi_3 = \langle 4, 1 \rangle$, the blackout interval for ϕ_3 can never exceed 3. Therefore, the general sbf for periodic models given in Equation (2.3) in Section 2.2.1 is pessimistic for our case. Improved sbf_ϕ is illustrated in Figure 7.7, and the corresponding schedule of ϕ is shown in Figure 7.6. This

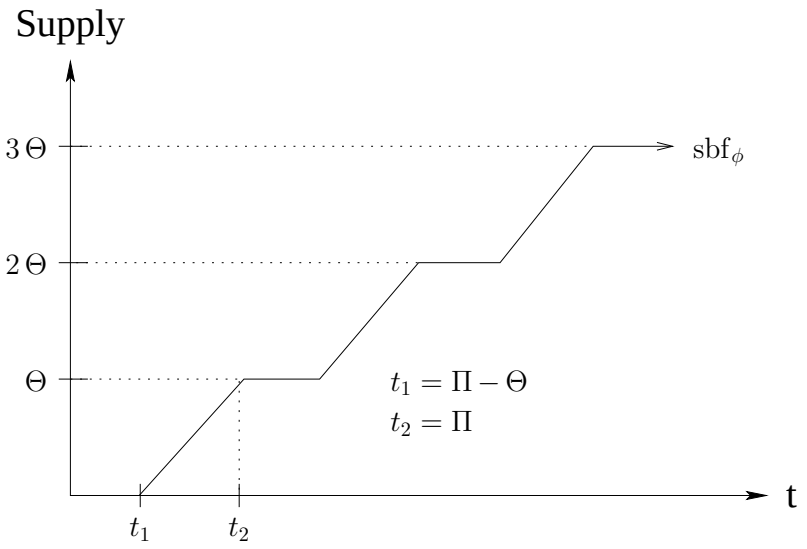


Figure 7.7: Improved sbf_ϕ for periodic resource model ϕ

improved sbf is defined as follows.

$$\text{sbf}_\phi(t) = \left\lfloor \frac{t}{\Pi} \right\rfloor \Theta + \max\left\{0, t - (\Pi - \Theta) - \left\lfloor \frac{t}{\Pi} \right\rfloor \Pi\right\} \tag{7.2}$$

Recall from the discussions in Section 5.3 that the blackout interval in sbf of EDP resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$ is $\Pi + \Delta - 2\,\Theta$. Since $\Delta \geq \Theta$, this blackout interval cannot be smaller than $\Pi - \Theta$ under any circumstance. But, as described above, $\Pi - \Theta$ is equal to the improved blackout interval of periodic resource models. Then, there will be no advantage in using EDP model based partition interfaces, and therefore we focus on periodic models in the remainder of this chapter.

7.3.3 Schedulability condition for partitions

In this section we extend Theorem 7.1 to handle processes with non-zero offset values. In particular, we present schedulability conditions for a partition $\mathcal{P} = \langle \{\tau_1, \ldots, \tau_n\}, \text{DM} \rangle$, scheduled using periodic resource model $\phi = \langle \Pi, \Theta \rangle$.

Request function

When processes have non-zero offsets, identifying the critical arrival pattern to compute rbf is a non-trivial task. It has been shown that this arrival pattern could occur anywhere in the interval $[0, \text{LCM}]$, where LCM denotes least common multiple of process periods (see [78]). As a result, no closed form expression for rbf is known in this case [3]. Therefore, we now introduce the *request function* ($\text{rf} : \Re \times \Re \rightarrow \Re$), which for a time interval $[t_1, t_2)$, gives the amount of resource requested by the partition in that interval. Since rf does not give the worst-case resource request for an interval length, it can be computed without knowledge of the critical arrival pattern. Equation (7.3) gives function rf for a process τ_i in $\mathcal{P}$, assuming processes have zero jitter.

$$\text{rf}_{\mathcal{P},i}(t_1, t_2) = \sum_{j=1}^{i} \left(\left\lceil \frac{t_2 - \text{O}_j}{\text{T}_j} \right\rceil - \left\lceil \frac{t_1 - \text{O}_j}{\text{T}_j} \right\rceil \right) \text{C}_j \tag{7.3}$$

When processes have non-zero jitter in addition to non-zero offsets, we must compute $\text{rf}_{\mathcal{P},i}$ assuming an arrival pattern that results in the maximum higher priority interference for τ_i. The following definition gives this arrival pattern for a job of τ_i with latest release time t, where $t = \text{O}_i + \text{J}_i + x\,\text{T}_i$ for some non-negative integer x.

Definition 7.1 (Arrival pattern with jitter [108]) *Recall that a job of process* $\tau = (\text{O}, \text{J}, \text{T}, \text{C}, \text{D})$ *is* dispatched *at time instant* $x\,\text{T} + \text{O}$ *for some non-negative integer* x*, and can be* released *for execution at any time in the interval* $[x\,\text{T} + \text{O}, x\,\text{T} + \text{O} + \text{J}]$*. Then, a job of* τ_i *with latest release time* t*, incurs maximum interference from higher priority processes in* $\mathcal{P}$ *whenever, (1) all higher priority processes with dispatch time before* t *are released at or before* t *with maximum jitter, and (2) all higher priority processes with dispatch time at or after* t *are released with zero jitter.*

[3] $\text{rbf}_{\mathcal{P},i}$ defined in Equation (7.1) is only an upper bound.

The request function for processes with non-zero offset and jitter values is then given by the following equation.

$$\mathrm{rf}_{\mathcal{P},i}(t_1, t_2) = \sum_{j=1}^{i} \left(\left\lceil \frac{t_2 - \mathrm{O}_j}{\mathrm{T}_j} \right\rceil - \left\lceil \frac{t_1 - \mathrm{O}_j - \mathrm{J}_j}{\mathrm{T}_j} \right\rceil \right) \mathrm{C}_j \tag{7.4}$$

Schedulability conditions

The following theorem presents exact schedulability conditions for partition $\mathcal{P}$ under periodic resource model ϕ.

Theorem 7.2 (Schedulability under DM (processes with offset and jitter)) *Let* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *denote a set of processes, where for each* i, $\tau_i = (\mathrm{O}_i, \mathrm{J}_i, \mathrm{T}_i, \mathrm{C}_i, \mathrm{D}_i)$. *Partition* $\mathcal{P} = \langle \mathcal{T}, \mathrm{DM} \rangle$ *is schedulable using a periodic resource model* $\phi = \langle \Pi, \Theta \rangle$ *iff* $\forall i : 1 \leq i \leq n, \forall t_x$ *s.t.* $t_x + \mathrm{D}_i - \mathrm{O}_i - \mathrm{J}_i < \mathrm{LCM}_{\mathcal{P}}$ *and* $t_x = \mathrm{O}_i + \mathrm{J}_i + x\,\mathrm{T}_i$ *for some non-negative integer* x, $\exists t \in (t_x, t_x + \mathrm{D}_i - \mathrm{O}_i - \mathrm{J}_i]$ *such that*

$$\mathrm{rf}_{\mathcal{P},i}(0, t) \leq \mathrm{sbf}_{\phi}(t) \text{ and } \mathrm{rf}_{\mathcal{P},i}(t_x, t) \leq \mathrm{sbf}_{\phi}(t - t_x) \tag{7.5}$$

$\mathrm{rf}_{\mathcal{P},i}$ *is given by Equation* (7.4) *and* sbf_{ϕ} *is given by Equation* (7.2). *Also,* $\mathrm{LCM}_{\mathcal{P}}$ *denotes the least common multiple of process periods* $\mathrm{T}_1, \ldots, \mathrm{T}_n$.

Proof To prove that these conditions are sufficient for schedulability of $\mathcal{P}$, we must validate the following statements: (1) it is sufficient to check schedulability of all jobs whose deadlines lie in the interval $[0, \mathrm{LCM}_{\mathcal{P}}]$, and (2) Equation (7.5) guarantees that the job of τ_i with latest release time t_x, is schedulable using periodic resource model ϕ.

Since $\mathrm{D}_i \leq \mathrm{T}_i$ and $\mathrm{O}_i \leq \mathrm{D}_i$ for all i, no process released before $\mathrm{LCM}_{\mathcal{P}}$ can execute beyond $\mathrm{LCM}_{\mathcal{P}}$ without violating its deadline. Furthermore, dispatch pattern of processes in $\mathcal{P}$ is periodic with period $\mathrm{LCM}_{\mathcal{P}}$. Therefore, it is sufficient to check the schedulability of all jobs in the interval $[0, \mathrm{LCM}_{\mathcal{P}}]$.

We now prove statement (2). Consider the job of τ_i with latest release time t_x. For this job to be schedulable under resource model ϕ, higher priority interference encountered by the job in interval $[t_x, t_x + t)$ must be satisfied by resource model ϕ. This higher priority interference arises from processes released before t_x, as well as from those released at or after t_x. Condition $\text{rf}_{\mathcal{P},i}(t_x, t) \leq \text{sbf}_\phi(t - t_x)$ guarantees that ϕ provides enough supply to satisfy the interference from processes released at or after t_x. To account for the interference from processes released before t_x, we have the second condition, *i.e.*, $\text{rf}_{\mathcal{P},i}(0, t) \leq \text{sbf}_\phi(t)$. This condition ensures that the minimum resource provided by ϕ in an interval of length t, is at least as much as the total higher priority interference up to time t. This proves that these conditions are sufficient for schedulability of partition $\mathcal{P}$.

We now prove that these conditions are also necessary for schedulability of $\mathcal{P}$. For this purpose, observe that $\text{rf}_{\mathcal{P},i}(0, t) \leq \text{sbf}_\phi(t)$ is a necessary condition to guarantee that resource model ϕ satisfies the higher priority interference in interval $[0, t)$. Furthermore, this condition alone is not sufficient, because it does not guarantee that ϕ will provide enough resource in interval $[t_x, t)$. The second condition ensures this property. □

Periodic resource model based interface for partition $\mathcal{P}$ can be generated using Theorem 7.2. Assuming period Π is equal to $\Pi_\mathcal{P}$, we can use this theorem to compute the smallest capacity Θ that guarantees schedulability of $\mathcal{P}$. When compared to Theorem 7.1, this theorem represents a computationally expensive (exponential versus pseudo-polynomial), but more accurate interface generation technique. In fact, for many avionics systems we expect this technique to be computationally efficient as well. For instance, if process periods are harmonic as in many avionics systems, then $\text{LCM}_\mathcal{P}$ is simply the largest process period, and our technique has pseudo-polynomial complexity in this case.

Although Theorem 7.2 presents an exact schedulability condition for $\mathcal{P}$, it ignores the preemption and blocking overheads incurred by processes in $\mathcal{P}$. Hence, in the following section, we extend our definition of rf to account for these overheads.

Blocking and preemption overheads

Recollect that processes incur blocking overhead because of mutual exclusion requirements modeled using semaphores. Blocking occurs when a lower priority process is executing in a critical section, and a higher priority process cannot preempt the lower priority process. In this case the higher priority process is said to be blocked by the lower priority process, resulting in blocking overheads. Assuming critical sections span entire processes, two properties of this overhead can be derived immediately: (1) this overhead varies with each job of a process, and (2) any job of a process can be blocked by at most one lower priority process.

Consider a process set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ and partition $\mathcal{P} = \langle \mathcal{T}, \text{DM} \rangle$. We now present an approach to bound the blocking overhead for a job of process τ_l released at time t. Specifically, we compute the bound when this job is blocked by some process having priority lower than that of τ_i, for some $i \geq l$. We assume that all processes with priority lower than τ_i can potentially block this job of τ_l. Our bound is given as

$$BO_{\mathcal{P},l,i}(t) = \max_{k \in [i+1,\ldots,n]} \{\min \{I_k, \mathrm{C}_k\}\}, \tag{7.6}$$

where I_k is defined as

$$I_k = \begin{cases} 0 & \left\lfloor \frac{t}{\mathrm{T}_k} \right\rfloor \mathrm{T}_k + \mathrm{O}_k \geq t \text{ or } \left\lfloor \frac{t}{\mathrm{T}_k} \right\rfloor \mathrm{T}_k + \mathrm{D}_k \leq t \\ \left\lfloor \frac{t}{\mathrm{T}_k} \right\rfloor \mathrm{T}_k + \mathrm{D}_k - t & \textit{Otherwise} \end{cases}$$

For each process τ_k, we compute its largest interference on the job of τ_l released at time t, and then choose the maximum over all τ_k that have priority lower than τ_i. Any such τ_k released at

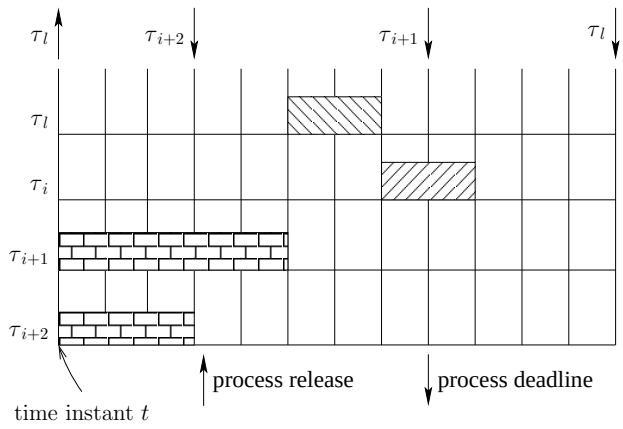


Figure 7.8: Illustrative example for blocking overhead $BO_{\mathcal{P},l,i}(t)$

or before t can block this job of τ_l, and this blocking overhead is at most its worst case execution time. Equation (7.6) uses this observation to compute the interference from τ_k. Figure 7.8 gives an illustrative example for this blocking overhead. Let the worst case execution requirement of processes τ_{i+1} and τ_{i+2}, shown in the figure, be 5 time units. Since the deadline of process τ_{i+1} is $t+8$, its interference on the job of τ_l released at t is at most 8. However, its worst case execution requirement is 5, and hence its interference is at most 5 time units. On the other hand, the deadline of process τ_{i+2} is $t+3$, and hence its maximum interference on this job of τ_l is 3 time units.

Note that Equation (7.6) only gives an upper bound, because the execution of processes τ_j, with $j \leq i$, could be such that no τ_k is able to execute before t. The following equation presents a quantity $BO_{\mathcal{P},l,i}(t_1, t_2)$, which bounds the blocking overhead incurred by all jobs of τ_l released in the interval $[t_1, t_2)$.

$$BO_{\mathcal{P},l,i}(t_1, t_2) = \sum_{t : t \in [t_1, t_2) \text{ and } \tau_l \text{ released at } t} BO_{\mathcal{P},l,i}(t) \tag{7.7}$$

When a higher priority process preempts a lower priority process, the context of the lower priority process must be stored for later use. When the lower priority process resumes its execution at some later time instant, this context must be restored. Thus, every preemption results in an execution overhead associated with storing and restoring of process contexts. Many different

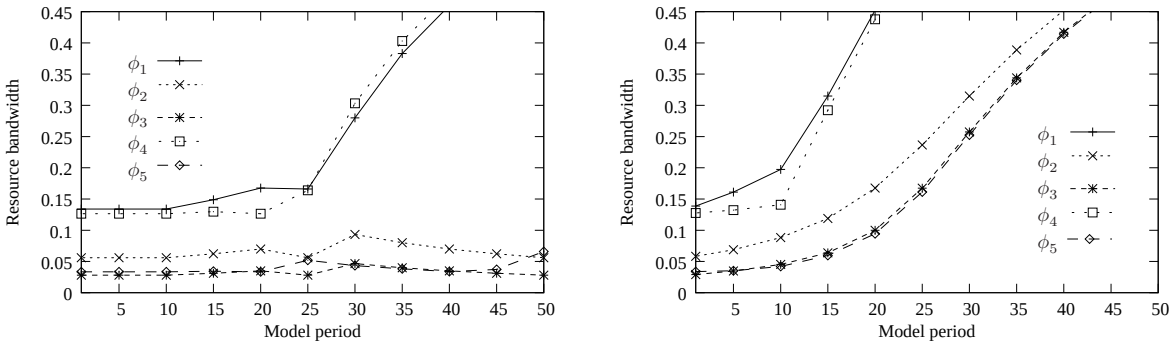
techniques for bounding this preemption overhead have been proposed in the past (see [35, 94, 53]). Under the assumption that every job of a process preempts a lower priority process, very loose upper bounds have been used in analysis of fixed priority schedulers [35]. Ramaprasad and Mueller [94] have proposed a tighter upper bound for processes scheduled under RM, and their technique can be extended to other fixed priority schedulers. However, they only present an algorithm to bound the preemptions, but do not give any closed form equations. Easwaran *et. al.* [53] have proposed an analytical upper bound for the number of preemptions under fixed priority schedulers. They presented these bounds for processes with non-zero offset values and zero jitter. These equations can be easily extended to account for jitter in process releases, as well as for blocking overheads. We assume that an upper bound on the number of preemptions is obtained using one such existing technique. Furthermore, we let $PO_{\mathcal{P},i}(t_1, t_2)$ denote this upper bound in the interval $[t_1, t_2)$, for preemptions incurred by processes that have priority at least as much as τ_i. Assuming δ_p denotes the execution overhead incurred by processes for each preemption, request function with blocking and preemption overheads is then given as

$$\mathrm{rf}_{\mathcal{P},i}(t_1, t_2) = \sum_{j=1}^{i} \left[\left(\left\lceil \frac{t_2 - \mathrm{O}_j}{\mathrm{T}_j} \right\rceil - \left\lceil \frac{t_1 - \mathrm{O}_j - \mathrm{J}_j}{\mathrm{T}_j} \right\rceil \right) \mathrm{C}_j + BO_{\mathcal{P},j,i}(t_1, t_2) \right] + PO_{\mathcal{P},i}(t_1, t_2)\delta_p \tag{7.8}$$

7.3.4 Interface generation for sample ARINC-653 workloads

We now demonstrate the effectiveness of our technique using sanitized data sets obtained from an avionics system. These data sets are specified in Appendix C. There are 7 workloads, where each workload represents a set of partitions scheduled on an uniprocessor platform having bandwidth one. We consider two types of workloads; workloads in which tasks have non-zero offsets but zero jitter (workloads 1 and 2 in Appendix C.1), and workloads in which tasks have non-zero jitter but zero offsets (workloads 3 thru 7 in Appendix C.2).

Each workload is specified using a xml schema, which can be described as follows. The top level tag <*system os-scheduler=*"DM"> identifies the system level scheduler under which the entire workload is scheduled. The next level tag <*component max-period=*"" *min-period=*""



(a) Using Theorem 7.2

(b) Using approach described in Section 2.2.3

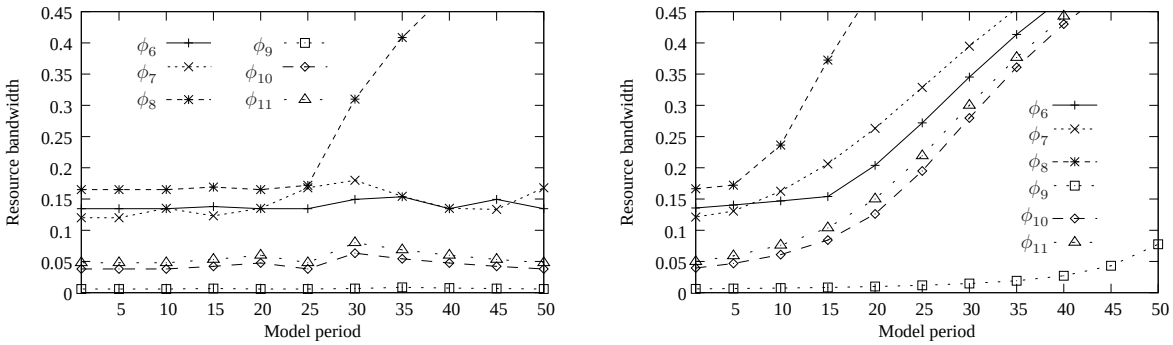
Figure 7.9: Interfaces for partitions P1, . . . , P5

scheduler="" name=""> identifies a partition in the workload. *min-period* and *max-period* define the range of values for interface period, *scheduler* defines the scheduling algorithm used by this partition, and *name* defines the name of the partition. The last level tag < *task offset="" jitter="" period="" capacity="" deadline="" />* defines a periodic task $\tau = (\mathrm{O}, \mathrm{J}, \mathrm{T}, \mathrm{C}, \mathrm{D})$. For workloads 1 and 2, tables in Appendix C.1 specify the total resource utilization of individual partitions ($\sum \frac{\mathrm{C}}{\mathrm{T}}$). Similarly, for workloads 3 thru 7, tables in Appendix C.2 specify the resource bandwidth reservations for individual partitions, in addition to total resource utilization. This bandwidth reservation is computed using the *vmips* field of the component tag in those workload specifications. Given a vmips value of x, the amount of resource bandwidth reserved is equal to $\frac{x}{17.76}$. These reservations are being used by system designers to allocate resource to partitions.

We have developed a tool set that takes as input hierarchical systems specified using the aforementioned xml schema, and generates as output resource model based component interfaces for them. In the following two sections we present our analysis results generated using this tool set.

Workloads with non-zero offsets

In this section, we consider workloads 1 and 2 specified in Appendix C.1. Firstly, we compare our technique with the existing compositional technique described in Section 2.2.3. We assume that



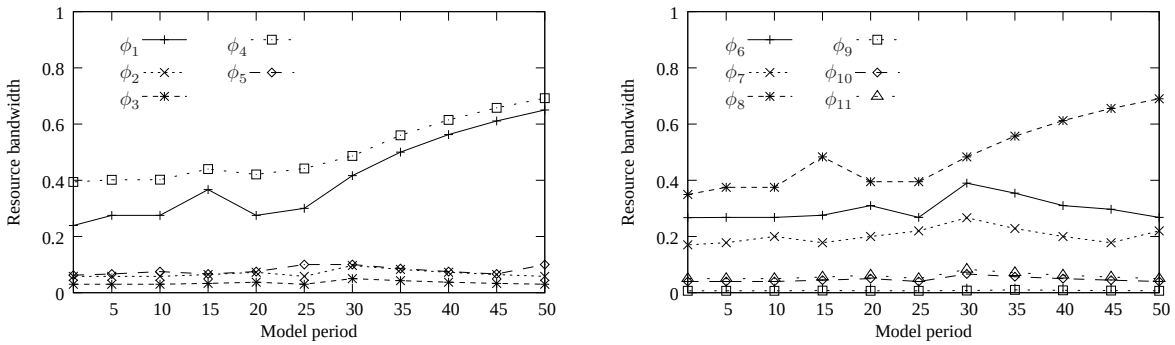
(a) Using Theorem 7.2

(b) Using approach described in Section 2.2.3

Figure 7.10: Interfaces for partitions P6, . . . , P11

this technique uses Theorem 7.1 to generate periodic resource model based partition interfaces, and therefore ignores process offsets. This approach does not account for preemption and blocking overheads incurred by processes. Hence to ensure a fair comparison, we ignore these overheads when computing interfaces using our approach as well. In Figures 7.9(a) and 7.10(a), we have plotted the resource bandwidths of interfaces obtained using our approach (Theorem 7.2). We have plotted these bandwidths for period values 1 and multiples of 5 up to 50. Note that since sbf_ϕ defined in Equation (7.2) is a linear function of capacity Θ, there is no need to use a linear lower bound like the one used in Section 2.2.3. Figure 7.9(a) shows the interfaces for partitions P1, . . . , P5, and Figure 7.10(a) shows the interfaces for partitions P6, . . . , P11. Similarly, we also obtained partition interfaces using Theorem 7.1 as discussed above, and their resource bandwidths are plotted in Figures 7.9(b) and 7.10(b).

As can be seen from these plots, interfaces obtained using our approach have a much smaller resource bandwidth when compared to those obtained using the existing technique. This gain in efficiency is because of two reasons: (1) we use a tighter sbf in Theorem 7.2 when compared to existing approach, and (2) existing approach ignores process offsets, and hence generates pessimistic interfaces. Although this is only an illustrative example, it is easy to see that the advantages of our interface generation technique hold in general. From the plots in Figures 7.9(a) and 7.10(a)



(a) Partitions P1, . . . , P5

(b) Partitions P6, . . . , P11

Figure 7.11: Partition interfaces with blocking and preemption overheads

we can also see that for some period values, bandwidths of our periodic resource models are equal to the utilization of corresponding partitions. Since utilization of a partition is the minimum possible bandwidth of a resource model that can schedule the partition, our approach generates optimal resource models for these periods. In these plots it can also be observed that the bandwidth increases sharply beyond a certain period. For interfaces ϕ_1, ϕ_4, and ϕ_8 corresponding to partitions $\mathcal{P}_1, \mathcal{P}_4$, and $\mathcal{P}_8$ respectively, the bandwidth increases sharply beyond period 25. This increase can be attributed to the fact that in these partitions the smallest process period is also 25. In our examples, since smallest process period corresponds to the earliest deadline in a partition, resource models with periods greater than this smallest value require larger bandwidth to schedule the partition.

Finally, we also generated partition interfaces using Theorem 7.2, taking into account preemption and blocking overheads. The resource bandwidth of these interfaces are plotted in Figures 7.11(a) and 7.11(b). For preemption overhead we assumed that the overhead for each preemption δ_p is 0.1, and that every job of a process preempts some lower priority process. Blocking overhead was computed using the upper bound given in Equation (7.7). As expected, resource bandwidths of these interfaces are significantly higher in comparison to the bandwidths in Figures 7.9(a) and 7.10(a) [4]. Since our preemption and blocking overheads are only upper bounds

[4]Y-axis in Figures 7.11(a) and 7.11(b) ranges from 0 to 1, whereas in Figures 7.9(a) and 7.10(a) it ranges from

and not necessarily tight, the minimum bandwidths of resource models that can schedule these partitions lie somewhere in between the two plots.

Workloads with non-zero jitter

In this section, we consider workloads 3 thru 7 specified in Appendix C.2. Since these workloads have zero offsets, we used Theorem 7.1 to generate periodic resource model based partition interfaces. In this theorem, we used sbf given by Equation (7.2), and interface periods are as specified by the *min-period* and *max-period* fields of component tags [5]. For preemption overheads we assumed that the overhead for each preemption δ_p is 0.1, and that every job of a process preempts some lower priority process. For blocking overheads we assumed that every lower priority process can block the process under consideration, up to its worst case execution time. Consider the process set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ and partition $\mathcal{P} = \langle \mathcal{T}, \text{DM} \rangle$. Then, for a process $\tau_l \in \mathcal{T}$, its blocking overhead is equal to $\max_{k>l}\{\text{C}_k\}$.

We now compare the bandwidth of generated interfaces with the reserved bandwidth specified by *vmips* field of component tags. Tables 7.1, 7.2, 7.3, 7.4, and 7.5, list the following four parameters for each partition in workloads 3 thru 7: (1) Total utilization of the partition ($\sum \frac{\text{C}}{\text{T}}$), (2) Reserved bandwidth ($\frac{vmips}{17.76}$), (3) Interface bandwidth computed using technique described above, and (4) Percentage increase in bandwidth ($\frac{\text{reserved} - \text{computed}}{\text{computed}} \times 100$). As can be seen from these tables, bandwidths of partition interfaces generated using our technique are significantly smaller than reserved bandwidths of partitions. However, when generating partition interfaces, we ignore the resource requirements of aperiodic processes in partitions. These aperiodic processes are identified by a *period* field equal to zero in the task tag. For example, they are present in partition "PART26 ID=26" of workload 4 and partition "PART22 ID=22" of workload 6. Since

0 to 0.45.

[5] Note that *min-period* = *max-period* in all the component tags in workloads 3 thru 7.

Partition name	Utilization	Reserved band.	Computed band.	Overhead
PART16 ID=16	0.01965	0.04505	0.0246	83.1%
PART29 ID=29	0.199415	0.37669	0.3735	0.9%
PART35 ID=35	0.05168	0.22185	0.0717	209.4%
PART20 ID=20	0.035125	0.09798	0.0589	66.3%
PART32 ID=32	0.033315	0.08164	0.0781	4.5%
PART36 ID=36	0.045	0.11036	0.12	−8%
PART33 ID=33	0.0379	0.09178	0.0579	58.5%
PART34 ID=34	0.04764	0.10755	0.0676	59.1%
PART17 ID=17	0.00408	0.01126	0.0082	37.3%
PART31 ID=31	0.00684	0.01689	0.0137	23.3%

Table 7.1: Resource bandwidth for workload 3

Partition name	Utilization	Reserved band.	Computed band.	Overhead
PART30 ID=30	0.11225	0.23086	0.169	36.6%
PART16 ID=16	0.01965	0.04505	0.0246	83.1%
PART20 ID=20	0.035125	0.09797	0.0589	66.3%
PART17 ID=17	0.00408	0.01126	0.0082	37.3%
PART26 ID=26	0.13496	0.44932	0.2538	77%
PART27 ID=27	0.02784	0.06869	0.0478	43.7%
PART28 ID=28	0.0552	0.12106	0.0752	61%

Table 7.2: Resource bandwidth for workload 4

the workloads do not specify any deadlines for these processes (they execute as background processes in ARINC-653), we cannot determine the resource utilization of these tasks. Then, one may argue that the difference in reserved bandwidth and bandwidth computed by our technique, is in fact used by aperiodic processes. Although this can be true, our results show that even for partitions with no aperiodic processes, there are significant resource savings using our technique.

Partition name	Utilization	Reserved band.	Computed band.	Overhead
PART15 ID=15	0.5208	0	0.5224	
PART13 ID=13	0.01126	0.03378	0.0163	107.2%
PART12 ID=12	0.0050	0.01126	0.02	−43.7%

Table 7.3: Resource bandwidth for workload 5

Partition name	Utilization	Reserved band.	Computed band.	Overhead
PART16 ID=16	0.01965	0.04505	0.0246	83.1%
PART19 ID=19	0.14008	0.32939	0.2284	44.2%
PART21 ID=21	0.12751	0.30011	0.2667	12.5%
PART22 ID=22	0.13477	0.31137	0.2631	18.3%
PART17 ID=17	0.00408	0.01126	0.0082	37.3%

Table 7.4: Resource bandwidth for workload 6

Partition name	Utilization	Reserved band.	Computed band.	Overhead
PART45 ID=45	0.00325	0.02815	0.01	181.5%

Table 7.5: Resource bandwidth for workload 7

7.4 Partition scheduling

In Section 7.3 we generated periodic resource model based interfaces for partitions. To schedule these interfaces on the uniprocessor platform, we must transform each resource model into a task that the higher level DM scheduler can use. In this section, we first describe one such transformation from periodic resource models to periodic tasks (processes), and then propose techniques to account for preemption overheads incurred by partitions.

7.4.1 Transformation of partition interfaces

Let the partition set $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be scheduled on an uniprocessor platform under DM scheduler. Furthermore, let each partition $\mathcal{P}_i$ be represented by a periodic resource model based interface $\phi_i = \langle \Pi_i, \Theta_i \rangle$. Without loss of generality we assume that $\Pi_1 \leq \ldots \leq \Pi_n$. Consider the transformation which for an interface ϕ_i generates the process $\tau_i = (0, 0, \Pi_i, \Theta_i, \Pi_i)$. Note that this transformation is identical to the transformation presented in Section 2.2.3. The following theorem observes that this transformation is both necessary and sufficient with respect to resource requirements of ϕ_i.

Theorem 7.3 *Let each partition interface $\phi_i = \langle \Pi_i, \Theta_i \rangle$ be transformed into a process $\tau_i = (0, 0, \Pi_i, \Theta_i, \Pi_i)$. Then for each i, the amount of resource required to schedule τ_i is at least as much as sbf_{ϕ_i}. Also, if the resource used by process τ_i in a schedule is at least as much as sbf_{ϕ_i}, then τ_i is schedulable.*

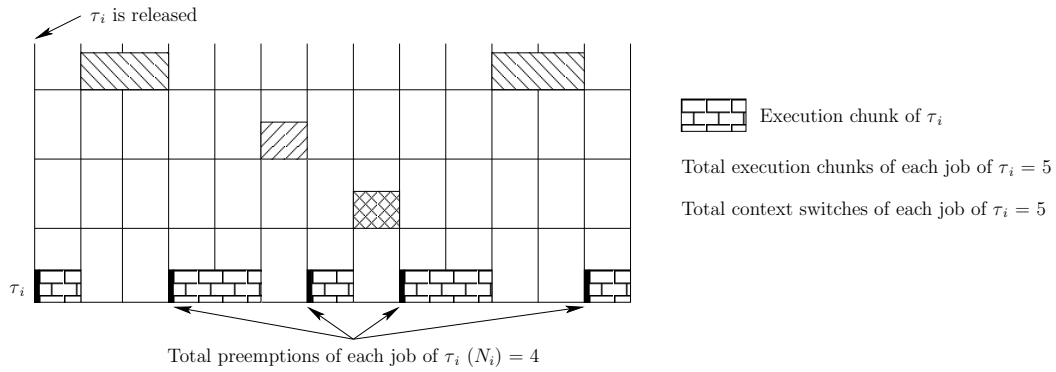


Figure 7.12: Preemption count terminology

Proof Direct from definition of sbf_{ϕ_i} in Equation (7.2). □

If each partition interface is transformed similarly, then processes in the resulting set $\{\tau_1, \ldots, \tau_n\}$ have implicit deadlines, zero offset values, and harmonic periods (partition periods are harmonic). Liu and Layland [82] have shown that DM is an optimal scheduler for such processes. In the following section we present a technique to count the number of preemptions incurred by this process set. The partition level schedule can then be generated after adjusting execution requirements of $\tau_1, \ldots, \tau_n$ to account for preemption overheads.

7.4.2 Partition level preemption overhead

Preemption overhead for partitions represented as processes, can be computed using the upper bounds described in Section 7.3. However, as described in the previous section, these processes are scheduled under DM, and have harmonic periods, implicit deadlines, and zero offset and jitter values. For such a process set, it is easy to see that every job of each process executes in the same time instants relative to its release time (see Figure 7.5). Therefore, every job of a process is preempted an identical number of times. For this case, we now develop an analytical technique to compute the exact number of preemptions.

Consider the process set $\tau_1, \ldots, \tau_n$ defined in the previous section. For each i, let N_i denote the number of preemptions incurred by each job of τ_i. We first give an upper bound for N_i,

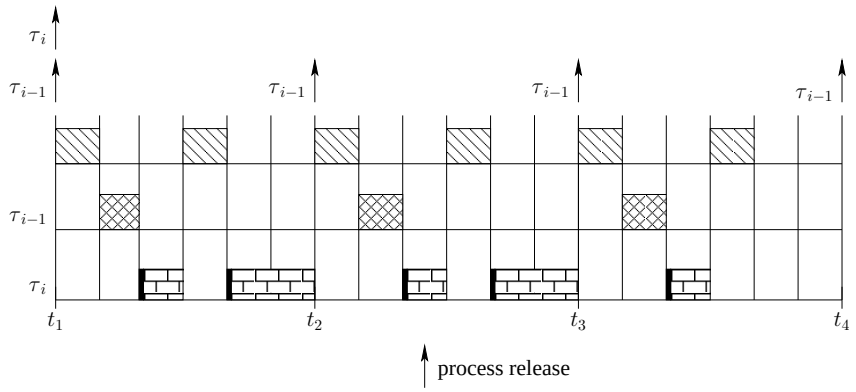


Figure 7.13: Overlapping execution windows of process τ_{i-1}

and later show how to tighten this bound. For this upper bound, we assume that the number of preemptions $N_1, \ldots, N_{i-1}$ for processes $\tau_1, \ldots, \tau_{i-1}$ respectively, are known. We also assume that the worst case execution requirements of these processes are adjusted to account for preemption overheads. Then, the following iterative equation gives an upper bound for N_i (see Theorem 7.4 for explanation of this equation).

$$N_i^{(k)} = \left\lceil \frac{\Theta_i^{(k)}}{\Pi_{i-1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} \Theta_j} \right\rceil \left(\frac{\Pi_{i-1}}{\Pi_1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} N_j \right) - 1 \tag{7.9}$$

In this equation we assume $\Theta_i^{(0)} = \Theta_i$ and $\Theta_i^{(k)} = \Theta_i + N_i^{(k-1)}\delta_p + \delta_p$, where δ_p denotes the execution overhead for each preemption. $N_i^{(k)}$ ignores the preemption incurred by process τ_i at the start of its execution, and hence the additional δ_p in capacity adjustment (see Figure 7.12). Then, the upper bound for N_i is given by that value of $N_i^{(k)}$ for which $N_i^{(k)} = N_i^{(k-1)}$.

Theorem 7.4 *Let N_i^* denote the value of $N_i^{(k)}$ in Equation* (7.9) *such that $N_i^{(k)} = N_i^{(k-1)}$. Then $N_i^* \geq N_i$.*

Proof In the k^{th} iteration, given $\Theta_i^{(k)}$, Equation (7.9) computes the number of dispatches of process τ_{i-1} that occur before the execution of $\Theta_i^{(k)}$ units of τ_i. For example, in Figure 7.13 there are three dispatches of τ_{i-1} that overlap with the execution of τ_i: intervals $(t_1, t_2], (t_2, t_3]$, and $(t_3, t_4]$. This computation is done inside the ceiling function by taking into account higher

priority interference for τ_i. We then determine the number of preemptions incurred by τ_i within the execution window of each these dispatches of τ_{i-1}. Since every job of a process executes in the same time instants relative to its release time, this number of preemptions is the same in each of these execution windows, except the first and last one. In the first window it is smaller by one because we ignore preemption at the start of execution of τ_i. In the last window it is smaller because execution of τ_i can terminate before the end of window. For example, in Figure 7.13 the number of preemptions incurred by τ_i in interval $(t_1, t_2]$ is 1, in interval $(t_2, t_3]$ is 2, and in interval $(t_3, t_4]$ is 1. Use of ceiling function implies that the first and last windows are treated similar to other execution windows, and this is one factor for the upper bound.

To determine the number of preemptions within each execution window of τ_{i-1}, Equation (7.9) computes the number of execution chunks of τ_i in each window. Each set of consecutive execution units of a process in a schedule is a single execution chunk (see Figure 7.12)[6]. The maximum possible number of chunks is given by $\frac{\Pi_{i-1}}{\Pi_1}$. However, since higher priority processes also execute in this window, τ_i does not necessarily have so many execution chunks. To get a tighter estimate for N_i, we subtract the execution chunks of higher priority processes from this maximum possible number. For each higher priority process τ_j, $\frac{\Pi_{i-1}}{\Pi_j}$ gives the number of jobs of τ_j in the current execution window, and N_j gives the number of preemptions incurred by each of those jobs. Then, the number of execution chunks of τ_j in the entire window is $(N_j + 1)\frac{\Pi_{i-1}}{\Pi_j}$. However, all of these execution chunks of τ_j cannot be always discarded; specifically the last one. Since the response time of τ_j need not necessarily coincide with a release of τ_1, τ_i could potentially continue its execution immediately after the last execution chunk of τ_j. For example, in Figure 7.14, τ_j's response time does not coincide with the release of τ_1, and hence τ_i can potentially execute in the marked time intervals. In Equation (7.9) we always use N_j for the number of execution chunks of τ_j, and hence the result is an upper bound. Finally, we subtract one from the entire number

[6]Note that the number of execution chunks is always one more than the number of preemptions encountered by the process.

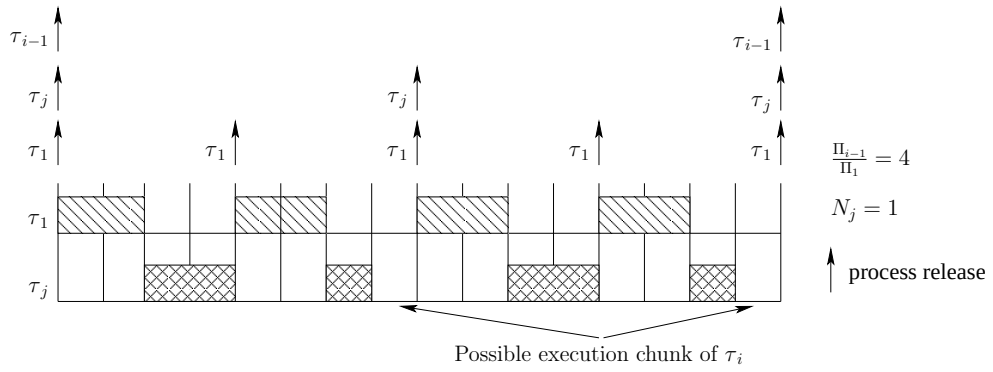


Figure 7.14: Execution chunks of higher priority process τ_j

to discount the preemption at the start of execution of τ_i. □

Since $\Theta_i^{(k)}$ is non-decreasing and cannot be greater than Π_i, this iterative computation must terminate and has pseudo-polynomial complexity. This computation only gives an upper bound for N_i due to two reasons: (1) the ceiling function, and (2) use of N_j as the count for execution chunks of process τ_j. In fact, Equation (7.9) cannot even be used to upper bound N_i, because it assumes knowledge of preemption counts $N_1, \ldots, N_{i-1}$. We now present a technique that overcomes these shortcomings. In particular, we modify Equation (7.9) as follows:

- We replace ceiling with the floor function, and add a separate expression that counts preemptions in the last execution window of τ_{i-1}.
- We replace N_j in the equation with a quantity I_j, which is either $N_j + 1$ or N_j, depending on whether the response time of τ_j coincides with a release of τ_1.

Let $N_i^{(k)'}$ denote the preemption count for τ_i in the last execution window of τ_{i-1}, when $\Theta_i^{(k)}$ is the execution requirement of τ_i. Then, N_i is given by the following iterative equation.

$$N_i^{(k)} = \left\lfloor \frac{\Theta_i^{(k)}}{\Pi_{i-1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} \Theta_j} \right\rfloor \left(\frac{\Pi_{i-1}}{\Pi_1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} I_j \right) + N_i^{(k)'} - 1 \tag{7.10}$$

In this equation we assume $\Theta_i^{(0)} = \Theta_i$ and $\Theta_i^{(k)} = \Theta_i + N_i^{(k-1)} \delta_p + \delta_p$. Also, N_i is given by that value of $N_i^{(k)}$ for which $N_i^{(k)} = N_i^{(k-1)}$. We now give equations to compute the two unknown

quantities, I_j and $N_i^{(k)'}$ in Equation (7.10).

$$I_j = \begin{cases} N_j + 1 & \left\lceil \frac{\mathrm{R}_j}{\Pi_1} \right\rceil = \left\lfloor \frac{\mathrm{R}_j}{\Pi_1} \right\rfloor \\ N_j & \text{Otherwise} \end{cases}$$

Here R_j denotes the worst case response time of process τ_j. Since $j \in [1, \ldots, i-1]$, N_j is known and therefore R_j can be computed. $N_i^{(k)'}$ is given by the following equation.

$$N_i^{(k)'} = \left\lceil \frac{\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}}{\Pi_1} \right\rceil - \sum_{j=2}^{i-1} \left\lceil \frac{\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}}{\Pi_j} \right\rceil I_j \tag{7.11}$$

In this equation $\mathrm{R}_i^{(k)}$ denotes the response time of τ_i with execution requirement $\Theta_i^{(k)}$, and $T_{i-1}^{(k)}$ is the time of last dispatch of τ_{i-1} (for example, t_3 in Figure 7.13).

$$T_{i-1}^{(k)} = \left\lfloor \frac{\Theta_i^{(k)}}{\Pi_{i-1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} \Theta_j} \right\rfloor \Pi_{i-1}$$

$\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}$ gives the total time taken by τ_i to execute in the last execution window of τ_{i-1}. This, along with the higher priority interference in the window, gives $N_i^{(k)'}$. The following theorem then observes that the preemption count generated using Equation (7.10) is equal to N_i.

Theorem 7.5 *Let N_i^* denote the value of $N_i^{(k)}$ in Equation* (7.10) *such that $N_i^{(k)} = N_i^{(k-1)}$. Then $N_i^* = N_i$.*

Proof Similar to the proof of Theorem 7.4. □

In this iterative procedure as well, $\Theta_i^{(k)}$ is non-decreasing and cannot be greater than Π_i. Therefore, the computation is of pseudo-polynomial complexity in the worst case. One may argue that the exact preemption count can also be obtained by simulating the execution of processes. Since process periods are harmonic, LCM is simply the largest process period, and therefore the simulation also runs in pseudo-polynomial time. However, in safety critical systems such as avionics, it is often required that we provide analytical guarantees for correctness. The iterative computation presented here serves this purpose.

Thus, each process τ_i can be modified to account for preemption overhead and is specified as $\tau_i = (0, 0, \Pi_i, \Theta_i + (N_i + 1)\delta_p, \Pi_i)$. If the resulting process set $\{\tau_1, \ldots, \tau_n\}$ is schedulable[7], then using Theorems 7.2, 7.3, and 7.5 we get that the underlying partitions can schedule their workloads.

Example 7.1 *As an example we consider workload 1 analyzed in Section 7.3.4. Partition interfaces for this workload are plotted in Figure 7.11(a). Let the chosen resource model periods for partitions P1, P2, P3, P4, and P5 be* $25, 50, 50, 25$, *and* 50 *respectively. Note that these periods are equal to the smallest process periods in partitions. Then, the corresponding resource model based interfaces are* $\phi_1 = \langle 25, 7.5\rangle, \phi_2 = \langle 50, 2.9\rangle, \phi_3 = \langle 50, 1.5\rangle, \phi_4 = \langle 25, 11.05\rangle$, *and* $\phi_5 = \langle 50, 5\rangle$. *From Section 7.4.1 we get that the corresponding processes are* $\tau_1 = (0, 0, 25, 7.5, 25), \tau_2 = (0, 0, 50, 2.9, 50), \tau_3 = (0, 0, 50, 1.5, 50), \tau_4 = (0, 0, 25, 11.05, 25)$, *and* $\tau_5 = (0, 0, 50, 5, 50)$. *Using Equation* (7.10) *we get that the number of preemptions for processes* τ_1, τ_2, τ_3, *and* τ_4 *are* 0 *and for process* τ_5 *is* 1. *Assuming* $\delta_p = 0.1$, *we get that the resulting process set is schedulable on an uniprocessor platform. This means that partitions P1,* $\ldots$, *P5 are schedulable.*

7.5 Conclusions

In this chapter we presented ARINC-653 standards for avionics real-time OS, and modeled it as a two level hierarchical system. We extended existing resource model based techniques to handle processes with non-zero offset values. We then used these techniques to generate partition level schedules. Design of real-time systems in modern day air-crafts is done manually through interactions between application vendors and system designers. Techniques presented in this chapter serve as a platform for principled design of partition level schedules. They also provide analytical correctness guarantees, which can be used in system certification.

[7]Liu and Layland [82] have given response time based schedulability conditions for this case.

Chapter 8

Virtual cluster-based scheduling on multiprocessors

8.1 Introduction

With rapid development in microprocessor technology, multiprocessor and multi-core designs are becoming an attractive solution to fulfill increasing performance demands. In the real-time systems community, there has been a growing interest in multiprocessor scheduling theories for elementary real-time systems. In general, existing approaches over m processors can fall into two categories: *partitioned* and *global* scheduling. Under partitioned scheduling each task is statically assigned to a single processor and is allowed to execute on that processor only. Under global scheduling tasks are allowed to dynamically migrate across m processors and execute on any of them.

In this chapter, we consider another approach using a notion of *processor cluster*. A cluster is a set of m' processors, where $1 \leq m' \leq m$. Under cluster-based scheduling, tasks are statically assigned to a cluster and then globally scheduled within the cluster. Cluster-based scheduling can be viewed as a generalization of partitioned and global scheduling; it is equivalent to partitioned

scheduling at one extreme end where we assign tasks to m clusters each of size one, and global scheduling at the other extreme end where we assign tasks to a single cluster of size m. Cluster-based scheduling can be further classified into two types: *physical* and *virtual* depending on how a cluster is mapped to processors in the platform. A physical cluster holds a static one-to-one mapping between its m' processors and some m' out of m processors in the platform [37]. A virtual cluster allows a dynamic one-to-many mapping between its m' processors and the m processors in the platform. Scheduling tasks in this virtual cluster can be viewed as scheduling them globally on all the m processors in the platform with amount of concurrency at most m'. A key difference is that physical clusters share no processors in the platform, while virtual clusters can share some.

Motivating example. We now illustrate the capabilities of cluster-based scheduling with an example. Consider a sporadic task system comprised of 6 tasks as follows: $\tau_1 = \tau_2 = \tau_3 = \tau_4 = (3, 2, 3)$, $\tau_5 = (6, 4, 6)$, and $\tau_6 = (6, 3, 6)$. The notation followed here is given in Section 2.1.1. Let this task set be scheduled on a multiprocessor platform comprised of 4 processors. It is easy to see that this task set is not schedulable under any partitioned scheduling algorithm, because no processor can be allocated more than one task. Figure 8.1 shows the schedule of this task set under global Earliest Deadline First (gEDF) [82], EDZL [39], Least Laxity First (gLLF) [88], FP-EDF [19], and US-EDF$\{m/(2m-1)\}$ [105] scheduling algorithms. As shown in the figure, the task set is not schedulable under any of these algorithms. Now consider cluster-based scheduling as follows: tasks τ_1, τ_2, and τ_3 are executed under gLLF on a cluster $\mathcal{C}_1$ comprised of 2 processors, and tasks τ_4, τ_5, and τ_6 are executed under gEDF on another cluster $\mathcal{C}_2$ comprised of 2 processors. The resulting schedule is shown in Figure 8.1, and as can be seen all the task deadlines are met.

In addition to being more general than physical clustering, virtual clustering is also less sensitive to task-processor mappings. This can be explained using the same example as above with an additional task $\tau_7 = (6, 1, 6)$. Just for comparison, suppose τ_7 is assigned to the first cluster $\mathcal{C}_1$ along with tasks τ_1, τ_2, and τ_3. Then, physical cluster-based scheduling cannot accommodate

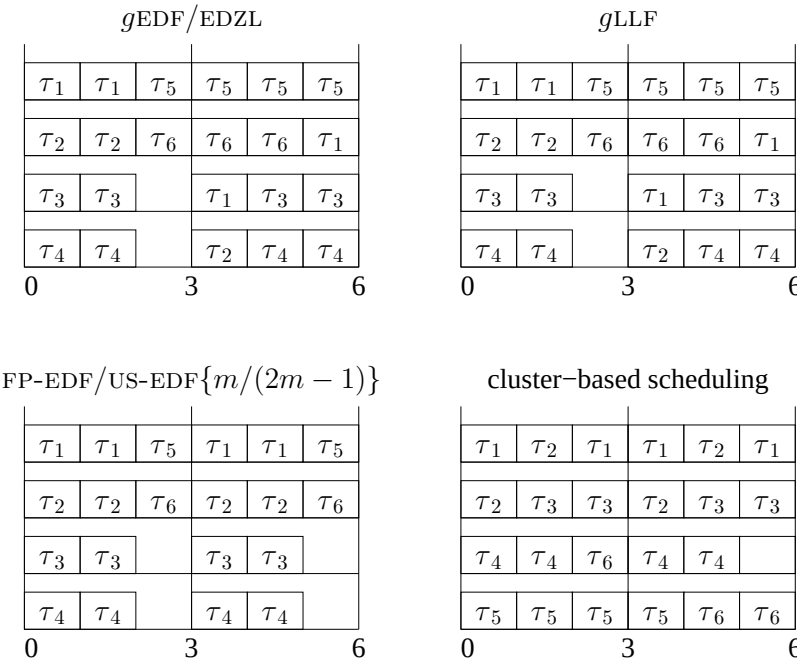


Figure 8.1: Motivating example

those two clusters on 4 processors. On the other hand, virtual clustering has a potential to accommodate them on 4 processors by dynamically re-allocating slack from cluster $\mathcal{C}_2$ to cluster $\mathcal{C}_1$ (time interval $(5, 6]$).

Clustering can also be useful as a mechanism to place a restriction on the amount of concurrency. Suppose m tasks can thrash a L2 cache in a multi-core platform, if they run in parallel at the same time. Then, one may consider allowing at most m' of the m tasks to run in parallel, in order to prevent them from thrashing the L2 cache. This can be easily done if the m tasks are assigned to a cluster of m' processors. A similar idea was used in [4].

Hierarchical scheduling. The notion of physical clustering requires intra-cluster scheduling only. This is because clusters are assigned disjoint physical processors, and hence tasks in different clusters cannot interfere with each others execution. However, the notion of virtual clustering inherently requires hierarchical scheduling; inter-cluster and intra-cluster scheduling. Under inter-cluster scheduling, physical processors are dynamically assigned to virtual clusters. Under intra-cluster scheduling, processor allocations given to a virtual cluster are assigned to tasks

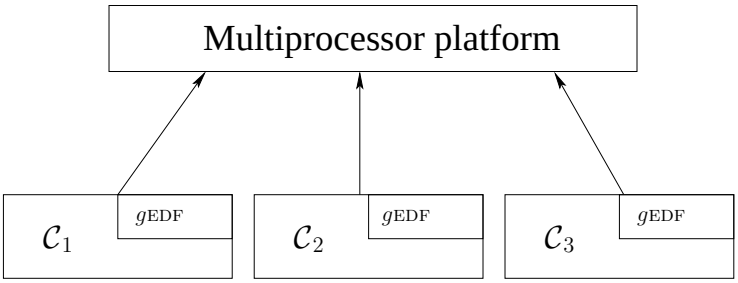


Figure 8.2: Example virtual clustering framework

in that cluster. Consider the example shown in Figure 8.2. Let a task set be divided into three clusters $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, each employing gEDF scheduling strategy. If we use physical clustering, then each cluster can be separately analyzed using existing techniques for gEDF. On the other hand if we use virtual clustering, then in addition to intra-cluster schedulability analysis, there is a need to develop techniques for scheduling clusters $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ on the multiprocessor platform. Therefore, supporting hierarchical multiprocessor scheduling is cardinal to the successful development of virtual clustering, and it is the main focus of this chapter.

There have been considerable studies on hierarchical uniprocessor scheduling as discussed in previous chapters. Denoting a collection of tasks and a scheduler as an *elementary component*, these studies employed the notion of a component interface to specify resources required for scheduling the component's tasks. Analogously, we denote a cluster along with tasks and scheduler assigned to it, as a *component* in hierarchical multiprocessor schedulers. To support inter-cluster scheduling, this chapter proposes a component interface that specifies resources required by tasks in the component's cluster. Inter-cluster scheduler can allocate processor supply to the cluster based on its interface. Intra-cluster scheduler can then use this processor supply to schedule tasks in the cluster. Many new issues arise to adopt the notion of a component interface from uniprocessor to multiprocessor scheduling. One of them is how to enable a component interface to carry information about concurrent execution of tasks in the component. For example, suppose a single task cannot execute in parallel. Then, multiple processors cannot be used concurrently

to satisfy the execution requirement of this single task. Such an issue needs to be handled for the successful development of component interfaces. In this chapter, we present one solution to this issue. Our approach is to capture in a component's interface, all the task-level concurrency constraints in that component. The interface demands enough processor supply from inter-cluster scheduler so that the intra-cluster scheduler can handle task-level concurrency constraints. As a result, the inter-cluster scheduler does not have to worry about this issue.

Contributions. The contributions of this chapter are five-fold. First, we introduce the notion of general hierarchical multiprocessor schedulers to support virtual cluster-based scheduling. Second, we present an approach to specify the task-level concurrency constraints in a component's interface. In Section 8.2 we introduce a multiprocessor resource model based interface that not only captures task-level concurrency constraints, but also specifies total resource requirements of the component. This enables the inter-cluster scheduler to schedule clusters using their interfaces alone. Third, since such interfaces represent partitioned resource supplies as opposed to dedicated ones, we also extend existing schedulability conditions for gEDF in this direction[1] (see Section 8.3). Such extensions to schedulability conditions are essential for supporting development of component interfaces. Fourth, we consider the optimization problem of minimizing the total resource requirements of component interface. In Section 8.4, we present an efficient solution to this problem based on the following property of our gEDF schedulability condition: total processor bandwidth required by a component interface to schedule tasks in the component increases, as number of processors allocated to the component's cluster increases. Thus, an optimal solution is obtained when we find the smallest number of processors that guarantee schedulability of the component. Fifth, in Section 8.5 we develop a overhead free inter-cluster scheduling framework based on McNaughton's algorithm [86]. Using this framework we present a new algorithm called

[1]We have chosen to focus on one scheduling algorithm in this chapter. However, the issues are same for other schedulers, and hence techniques developed here are applicable to other schedulers as well.

VC-IDT, for scheduling implicit deadline sporadic task systems on identical, unit-capacity multiprocessor platforms. We show that VC-IDT is an optimal scheduling algorithm[2], and to the best of our knowledge, this is the first such algorithm that does not satisfy the property of P-fairness [25] or ER-fairness [5]. We also use this framework to improve the resource utilization bound of US-EDF$\{m/(2m-1)\}$ algorithm.

8.2 Task and resource models

In this chapter we assume that tasks in elementary real-time systems are specified using the constrained deadline sporadic task model described in Section 2.1.1. We also assume that they are scheduled on the multiprocessor platform described in Section 2.1.2. For each job of a sporadic task $\tau = (\mathrm{T}, \mathrm{C}, \mathrm{D})$, we require that the C resource units be supplied non-concurrently to the job. This restriction is useful in modeling many real-world systems, because in general, all portions of a software program cannot be parallelized. We also assume that a job can be preempted on one processor and may resume execution on another processor with negligible preemption and migration overheads, as in the standard literature of global scheduling [62, 14, 32, 20]. We assume such a global scheduling strategy within each cluster, and in particular, we assume that tasks are scheduled using GEDF described in Section 2.1.3. In the following section we introduce multiprocessor resource models which we use as component interfaces.

8.2.1 Multiprocessor resource model

Periodic [80, 102], EDP in Chapter 5, bounded delay [56], etc., are examples of resource models that have been extensively used for analysis of hierarchical uniprocessor schedulers. These resource models can also be used as component interfaces in hierarchical multiprocessor schedulers. One

[2]If a set of implicit deadline sporadic tasks is schedulable on m identical, unit-capacity processors under some scheduling algorithm, then VC-IDT can also schedule it on m identical, unit-capacity processors.

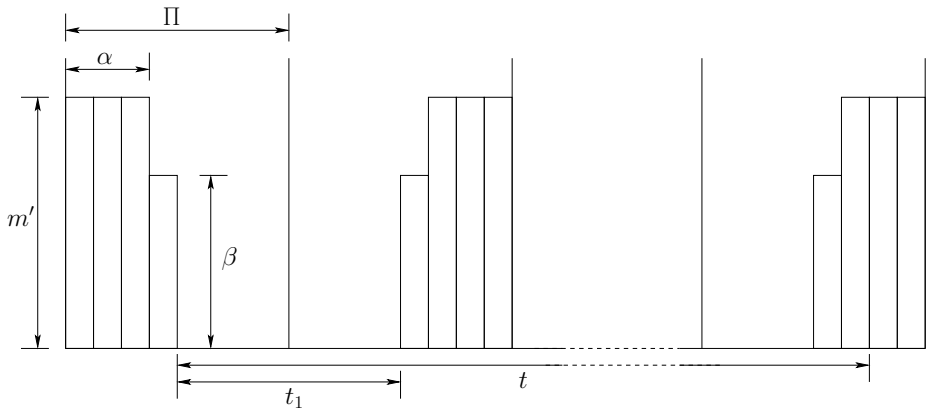


Figure 8.3: Schedule of μ w.r.t $\text{sbf}_\mu(t)$

way to achieve this is to consider m' identical resource models as a component interface, where m' is number of processors allocated to the component's cluster. However, this interface is restrictive because each processor contributes same amount of resource to the component as any other processor in the cluster. It is desirable to be more flexible, in that interfaces should be able to represent the collective resource requirements of clusters, without fixing the contribution of each processor a priori. Apart from increased flexibility, such interfaces can also improve resource utilization in the system.

We now introduce a *multiprocessor resource* model that specifies the characteristics of resource supply provided by an identical, unit-capacity multiprocessor platform. This resource model does not fix the contribution of each processor a priori, and hence is a suitable candidate for cluster interfaces.

Definition 8.1 (Multiprocessor periodic resource model (MPR)) *A* multiprocessor periodic resource *(MPR) model* $\mu = \langle \Pi, \Theta, m' \rangle$ *specifies that an identical, unit-capacity multiprocessor platform collectively provides* Θ *units of resource in every* Π *time units to a cluster comprising of* m' *processors. These* Θ *resource units are supplied with concurrency at most* m', i.e., *at any time instant at most* m' *physical processors are allocated to this resource model. It is then easy to see that a* feasible MPR model *must satisfy the condition* $\Theta \leq m' \Pi$. *Also,* $\frac{\Theta}{\Pi}$ *denotes the resource*

bandwidth of model μ.

As discussed in Section 2.2.1, the supply bound function (sbf) of a resource model lower bounds the amount of resource supply that the model guarantees in a given time interval. In uniprocessor systems, sbf is used in schedulability conditions to generate resource model based component interfaces (Sections 2.2.3 and 5.4). Extending this approach to multiprocessors, in this chapter we derive similar schedulability conditions to generate MPR model based component interfaces. Hence, we now present the sbf for a MPR model $\mu = \langle \Pi, \Theta, m' \rangle$. Figure 8.3 shows the schedule for μ that generates this minimum supply in a time interval of duration t, where $\alpha = \lfloor \frac{\Theta}{m'} \rfloor$ and $\beta = \Theta - m'\alpha$. As can be seen, length of the largest time interval with no supply is equal to $2\,\Pi - 2\left\lceil \frac{\Theta}{m'} \right\rceil$ (duration t_1 in the figure). Thereafter, μ is guaranteed to provide Θ units of resource in every Π time units. sbf_μ is then given by the following equation in which $k = \left\lfloor \frac{t - \left(\Pi - \left\lceil \frac{\Theta}{m'} \right\rceil\right)}{\Pi} \right\rfloor$. This function is also plotted in Figure 8.4.

$$\text{sbf}_\mu(t) = \begin{cases} k\,\Theta + \max\left\{0, \left[t - 2\,\Pi + \left\lceil \frac{\Theta}{m'} \right\rceil - k\,\Pi\right] m' + \Theta\right\} & t \geq \Pi - \left\lceil \frac{\Theta}{m'} \right\rceil \\ 0 & \text{Otherwise} \end{cases} \tag{8.1}$$

Note by setting $m' = 1$ in Equation (8.1), we get the sbf of periodic resource model $\langle \Pi, \Theta \rangle$ given in Section 2.2.1. This indicates that MPR models generalize periodic resource models. In uniprocessor systems, although schedulability conditions with sbf have been derived, a linear approximation of sbf is often used to improve the efficiency of interface generation. Hence, in anticipation, we present the following linear lower bound for sbf_μ. This function is also plotted in Figure 8.4.

$$\text{lsbf}_\mu(t) = \frac{\Theta}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta}{m'}\right)\right) \tag{8.2}$$

Uniprocessor resource models, such as periodic or EDP, allow a view that a component executes over an exclusive share of a physical uniprocessor platform. Extending this notion, MPR models

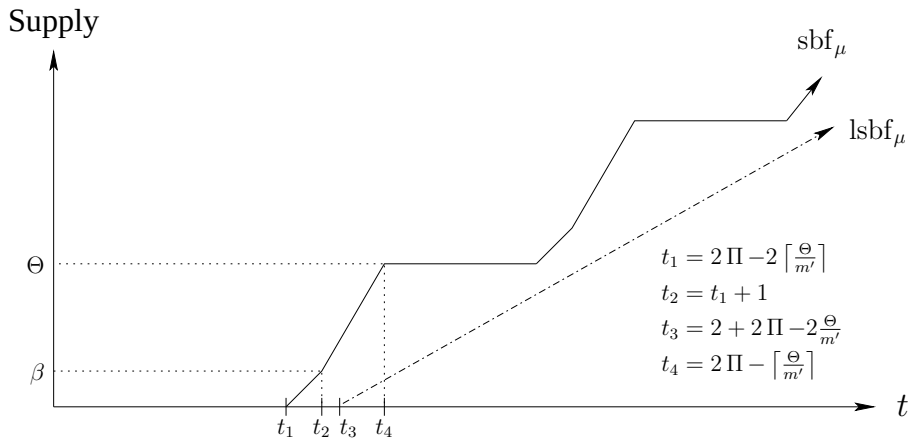


Figure 8.4: sbf$_\mu$ and lsbf$_\mu$

allow a view that a component, and hence the corresponding cluster, executes over an exclusive share of a physical multiprocessor platform. Although this view guarantees a minimum total resource share given by sbf, it does not enforce any distribution of this share over the processors in the platform apart from concurrency bound m'. In this regard MPR models are general, and hence our candidate for component interfaces.

8.3 Component schedulability condition

In this section we develop a schedulability condition for components in hierarchical multiprocessor schedulers, such that this condition accommodates the notion of a partitioned resource supply. Specifically, we extend existing gEDF schedulability conditions for dedicated resource, with the supply bound function of a MPR model. Any MPR model that satisfies this condition, can be used as an interface for the component.

We consider a component comprising of cluster $\mathcal{C}$ and tasks $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ scheduled under gEDF. To keep the presentation simple, we use notation $\mathcal{C}$ to refer to the component as well. We now develop a schedulability condition for $\mathcal{C}$ assuming it is scheduled using MPR model $\mu = \langle \Pi, \Theta, m' \rangle$, where m' denotes number of processors in the cluster. This condition uses the total processor demand of task set $\mathcal{T}$ for a given time interval. Existing

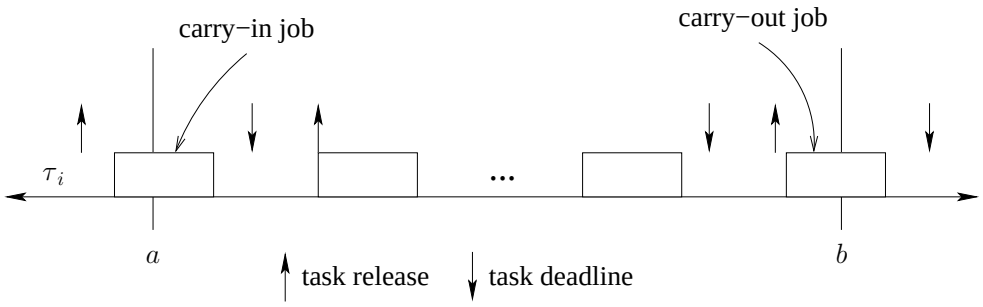


Figure 8.5: Workload of task τ_i in interval $[a, b]$

study [32] has developed an upper bound for this demand which we can use. Only upper bounds are known for this demand, because unlike the synchronous arrival sequence in uniprocessors, no notion of worst-case arrival sequence is known for multiprocessors [20]. Hence, we first summarize this existing upper bound for component demand, and then present our schedulability condition.

8.3.1 Component demand

Workload. The workload of a task τ_i in an interval $[a, b]$ gives the cumulative length of all intervals in which τ_i is executing, when task set $\mathcal{T}$ is scheduled under $\mathcal{C}$'s scheduler. This workload consists of three parts (illustrated in Figure 8.5): (1) the *carry-in* demand generated by a job of τ_i that is released prior to a, but did not finish its execution requirement until a, (2) the demand of a set of jobs of τ_i that are both released and have their deadlines within the interval, and (3) the *carry-out* demand generated by a job of τ_i that is released in the interval $[a, b)$, but does not finish its execution requirement until b.

Workload upper bound for τ_i under gEDF. If workload in an interval $[a, b]$ can be efficiently computed for all $a, b \geq 0$ and for all tasks τ_i, then we can obtain the exact demand of task set $\mathcal{T}$ in all intervals. However, since no such technique is known (apart from task set simulation), we use an upper bound for this workload obtained by Bertogna *et. al.* [32]. This bound is obtained under two assumptions: (1) some job of some task τ_k has a deadline at time instant b, and (2) this job of τ_k misses its deadline. In the schedulability conditions we develop, these assumptions hold for all time instants b that are considered. Hence, this is a useful bound

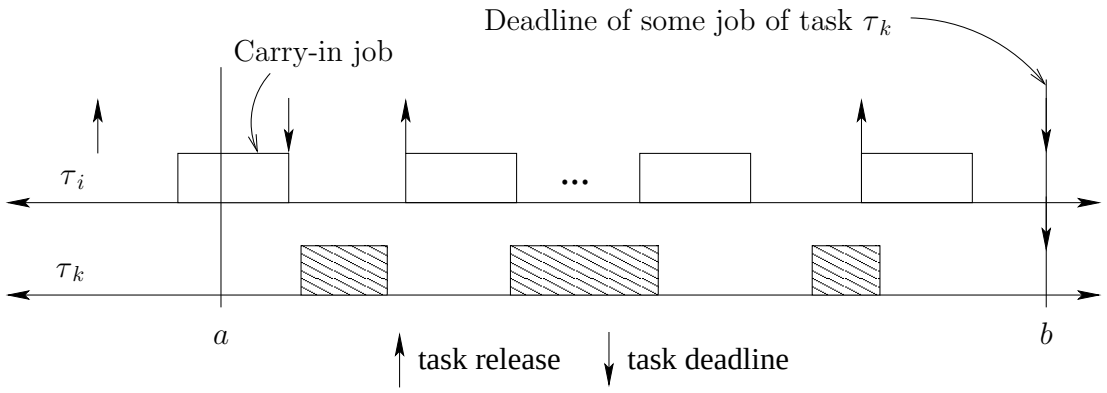


Figure 8.6: Dispatch and execution pattern of task τ_i for $\mathcal{W}_i(b-a)$

and we present it here. Figure 8.6 illustrates the dispatch pattern corresponding to this bound. A job of task τ_i has a deadline that coincides with time instant b. Jobs of τ_i that are released prior to time b are assumed to be released as late as possible. Also, the job of τ_i that is released before a, but has a deadline in interval $[a, b]$, is assumed to execute as late as possible. This imposes maximum possible interference on the job of τ_k with deadline at b. Let $\mathcal{W}_i(t)$ denote this workload bound for τ_i in a time interval of length $t(= b-a)$. Also, let $CI_i(t)$ denote the carry-in demand generated by the execution pattern shown in Figure 8.6. Then,

$$\mathcal{W}_i(t) = \left\lfloor \frac{t + (\mathrm{T}_i - \mathrm{D}_i)}{\mathrm{T}_i} \right\rfloor C_i + CI_i(t), \text{ where } CI_i(t) = \min\left\{ \mathrm{C}_i, \max\left\{ 0, t - \left\lfloor \frac{t + (\mathrm{T}_i - \mathrm{D}_i)}{\mathrm{T}_i} \right\rfloor \mathrm{T}_i \right\} \right\} \quad (8.3)$$

It has been shown that the actual workload of τ_i can never exceed $\mathcal{W}_i(b-a)$ in interval $[a, b]$, provided tasks are scheduled under gEDF and a deadline miss occurs for that job of τ_k whose deadline is b [32]. This follows from the observation that no job of τ_i with deadline greater than b can execute in interval $[a, b]$. In the following section we develop a schedulability condition for $\mathcal{C}$ using this workload bound.

8.3.2 Schedulability condition

We now present a schedulability condition for component $\mathcal{C}$ when it is scheduled using MPR model $\mu = \langle \Pi, \Theta, m' \rangle$. For this purpose, we extend an existing condition that checks schedulability of $\mathcal{C}$ on a dedicated resource, with the notion of sbf_μ.

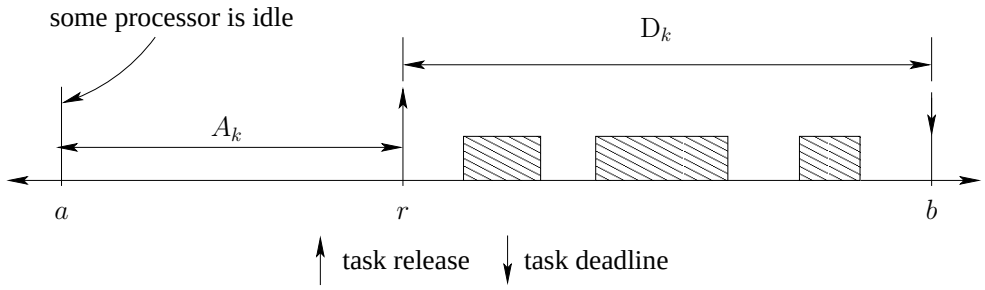


Figure 8.7: Example time instant a under dedicated resource

When task τ_k is scheduled on a dedicated resource comprised of m' unit-capacity processors, previous work identifies different time intervals that must be checked to guarantee schedulability of τ_k [20]. In particular, it assumes b denotes the missed deadline of some job of task τ_k (henceforth denoted as job τ_k^b), and then specifies different values of a, corresponding to interval $[a, b]$, that need to be considered. Figure 8.7 gives one such time instant a. It corresponds to a point in time such that, (1) at least one of the m' processors is idle at that instant, (2) it is prior to the release time of job τ_k^b (r in the figure), and (3) no processor is idle in the interval $(a, r]$. Observe that at each such time instant a, there can be at most $m' - 1$ tasks that contribute towards carry-in demand. This is because at most $m' - 1$ processors are executing jobs at a. Baruah uses this observation to come up with a schedulability condition in the dedicated resource case. Intuitively, this technique aims to derive a condition on the total higher priority workload in interval $[a, b]$ that guarantees a deadline miss for τ_k^b. In the following discussion, we extend this notion of time instant a for the case when τ_k is scheduled using MPR model μ.

When task τ_k is scheduled using MPR model μ, we denote a time instant as t_{idle} if at least one of the m' processors is idle at that instant, even though it is available for use as per supply μ. Figure 8.8 illustrates one such time instant, where r denotes the release time of job τ_k^b, A_k denotes the length of interval $(a, r]$, and $A_k + D_k$ denotes the length of interval $(a, b]$. Also, the empty boxes in that figure denote processor supply based on μ, and the striped boxes denote supply used by cluster $\mathcal{C}$. To check schedulability of task τ_k, we consider all time instants a such

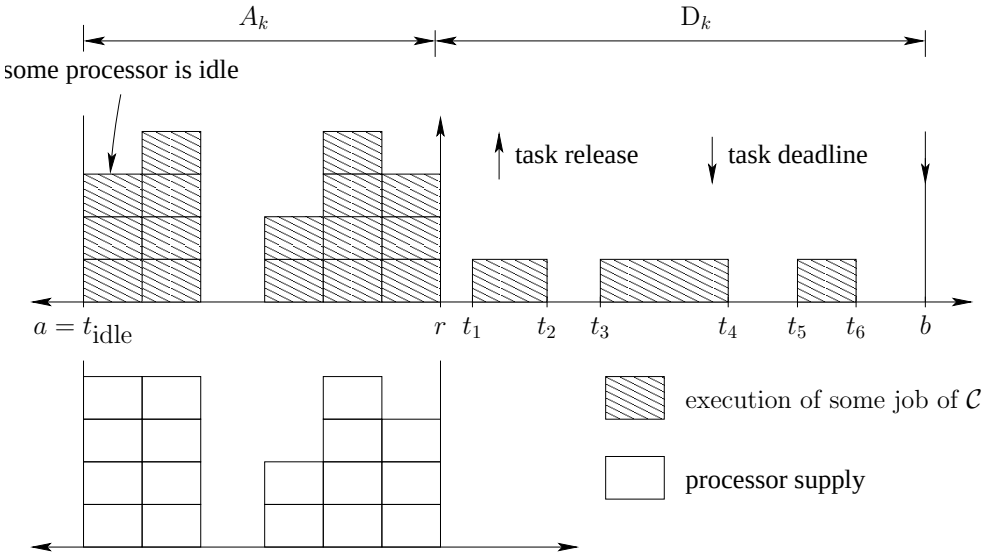


Figure 8.8: Example time instant t_{idle}

that: (1) a is t_{idle}, (2) $a \leq r$, and (3) no time instant in the interval $(a, r]$ is t_{idle}. The time instant illustrated in Figure 8.8 also satisfies these properties.

To derive schedulability condition for component $\mathcal{C}$, we consider all intervals $[a, b]$ as explained above, and derive conditions under which a deadline miss occurs for job τ_k^b. If τ_k^b misses its deadline, then the total workload of jobs having priority at least τ_k^b, must be greater than the total processor supply available to $\mathcal{C}$ in interval $[a, b]$. Let for each i, I_i denote the total workload in interval $[a, b]$ of jobs of τ_i that have priority at least τ_k^b. Then, since $\mathrm{sbf}_\mu(b-a)$ denotes a lower bound on the processor supply available to $\mathcal{C}$ in interval $[a, b]$, whenever τ_k^b misses its deadline it must be true that

$$\sum_{i=1}^{n} I_i > \mathrm{sbf}_\mu(b - a) = \mathrm{sbf}_\mu(A_k + \mathrm{D}_k) \tag{8.4}$$

This inequality follows from the following observations: (1) the actual processor supply available to component $\mathcal{C}$ in interval $[a, b]$ is at least $\mathrm{sbf}_\mu(A_k + \mathrm{D}_k)$, and (2) there are no t_{idle} time instants in interval (a, b), *i.e.*, all available processor supply is used by $\mathcal{C}$ to schedule tasks from $\mathcal{T}$. For $\mathcal{C}$ to be schedulable using μ, it then suffices to show that for all tasks τ_k and for all values of A_k, Equation (8.4) is invalid.

We now derive an upper bound for each workload I_i. We separately consider the workload of τ_i in the following two interval classes: (1) time intervals in $[a, b]$ in which τ_k^b executes (intervals $[t_1, t_2], [t_3, t_4]$, and $[t_5, t_6]$ in Figure 8.8), and (2) the other time intervals in $[a, b]$. Let $I_{i,1}$ denote the workload of τ_i in intervals of type (1) and $I_{i,2}$ denote the workload of τ_i in intervals of type (2). We bound I_i using upper bounds for $I_{i,1}$ and $I_{i,2}$. Under dedicated resource, Baruah [20] only considered intervals of type (2) when deriving the schedulability condition. However, sbf of MPR models are only defined over contiguous time intervals. Hence, under partitioned resource we are required to consider the contiguous time interval obtained from the union of intervals of types (1) and (2).

Since the cumulative length of intervals of type (1) is at most C_k and there are at most m' processors on which $\mathcal{C}$ executes, the total workload of all tasks in intervals of type (1) is clearly upper bounded by $m'\,\mathrm{C}_k$. Therefore, $\sum_{i=1}^{n} I_{i,1} \leq m'\,\mathrm{C}_k$. To bound $I_{i,2}$ we use the workload upper bound $\mathcal{W}_i$ presented in Section 8.3.1. Recall that $\mathcal{W}_i(b-a)$ $(= \mathcal{W}_i(A_k + \mathrm{D}_k))$ upper bounds the workload of all jobs of τ_i that execute in interval $[a, b]$ and have priority higher than τ_k^b. Therefore, $\mathcal{W}_i(A_k + D_k)$ also upper bounds $I_{i,2}$. Furthermore, there is no need for $I_{i,2}$ to be larger than $A_k + \mathrm{D}_k - \mathrm{C}_k$, because we have already considered a total length of C_k for intervals of type (1). Also, when $i = k$, this bound can be further tightened, because in $I_{k,2}$ we do not consider the execution units of τ_k^b. These execution units are considered for intervals of type (1). Thus, we can subtract C_k from $\mathcal{W}_k(A_k + \mathrm{D}_k)$ and $I_{k,2}$ cannot be greater than A_k.

$$I_{i,2} \leq \bar{I}_{i,2} = \min\{\mathcal{W}_i(A_k + \mathrm{D}_k), A_k + \mathrm{D}_k - \mathrm{C}_k\} \qquad i \neq k$$

$$I_{k,2} \leq \bar{I}_{k,2} = \min\{\mathcal{W}_k(A_k + \mathrm{D}_k) - \mathrm{C}_k, A_k\}$$

Now by definition of time instant a, at most $m' - 1$ tasks can be active, and hence have carry-in demand, at a. This follows from the fact that at least one processor is not being used by $\mathcal{C}$ at time a, even though that processor is available as per supply μ. Hence, we only need to consider

$m'-1$ largest values of CI_i when computing an upper bound for $\sum_{i=1}^{n} I_{i,2}$ using above equations, where CI_i denotes the carry-in demand in $\mathcal{W}_i$. Let us now define the following two terms.

$$\hat{I}_{i,2} = \min\{\mathcal{W}_i(A_k + \mathrm{D}_k) - CI_i(A_k + \mathrm{D}_k), A_k + \mathrm{D}_k - \mathrm{C}_k\} \text{for all } i \neq k$$

$$\hat{I}_{k,2} = \min\{\mathcal{W}_k(A_k + \mathrm{D}_k) - \mathrm{C}_k - CI_k(A_k + \mathrm{D}_k), A_k\}$$

Let $L_{(m'-1)}$ denote a set of task indices such that if $i \in L_{(m'-1)}$, then $(\bar{I}_{i,2} - \hat{I}_{i,2})$ is one of the $m'-1$ largest values among all tasks. Then, the upper bound on worst-case resource demand in the interval $(a, b]$ can be defined as,

$$\mathrm{DEM}(A_k + \mathrm{D}_k, m') = \sum_{i=1}^{n} \hat{I}_{i,2} + \sum_{i:i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + m'\,\mathrm{C}_k\,.$$

The following theorem gives our schedulability condition, and its proof follows from the above discussions.

Theorem 8.1 *A component comprising of cluster $\mathcal{C}$ with m' processors and sporadic tasks $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$, is schedulable under g*EDF *using MPR model $\mu = \langle \Pi, \Theta, m' \rangle$, if for all tasks $\tau_k \in \mathcal{T}$ and all $A_k \geq 0$,*

$$\mathrm{DEM}(A_k + \mathrm{D}_k, m') \leq \mathrm{sbf}_{\mu}(A_k + \mathrm{D}_k). \tag{8.5}$$

In Theorem 8.1, if we set $\Theta = m'\,\Pi$ then we get the schedulability condition for dedicated resource proposed by Baruah [20]. This shows that our schedulability condition is no more pessimistic than the one proposed by Baruah. Although this theorem gives a schedulability test for component $\mathcal{C}$, it would be highly inefficient if we were required to check for all values of A_k. The following theorem shows that this is not the case.

Theorem 8.2 *If Equation* (8.5) *is violated for some A_k, then it must also be violated for a value satisfying the condition*

$$A_k < \frac{\mathrm{C}_{\Sigma} + m'\,\mathrm{C}_k - \mathrm{D}_k\left(\frac{\Theta}{\Pi} - U_{\mathcal{T}}\right) + U + B}{\frac{\Theta}{\Pi} - U_{\mathcal{T}}},$$

where C_Σ *denotes the sum of* $m'-1$ *largest* C_i*'s,* $U_\mathcal{T} = \sum_{i=1}^{n} \frac{\mathrm{C}_i}{\mathrm{T}_i}$*,* $U = \sum_{i=1}^{n} (\mathrm{T}_i - \mathrm{D}_i)\frac{\mathrm{C}_i}{\mathrm{T}_i}$ *and* $B = \Theta\left(2 - \frac{2\Theta}{m'\Pi}\right)$.

Proof It is easy to see that $\hat{I}_{i,2} \leq \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k)$ and $\bar{I}_{i,2} \leq \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k) + \mathrm{C}_i$, where dbf_{τ_i} is given by Definition 2.6 in Section 2.1.5. Then, the LHS of Equation (8.5) is $\leq \mathrm{C}_\Sigma + m'\,\mathrm{C}_k + \sum_{i=1}^{n} \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k)$. For this equation to be violated, it must be true that

$$\mathrm{C}_\Sigma + m'\,\mathrm{C}_k + \sum_{i=1}^{n} \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k) > \mathrm{sbf}_\mu(A_k + \mathrm{D}_k)$$

$$\mathrm{C}_\Sigma + m'\,\mathrm{C}_k + (A_k + \mathrm{D}_k)U_\mathcal{T} + U > \mathrm{sbf}_\mu(A_k + \mathrm{D}_k) \qquad \text{Using } \mathrm{dbf}_{\tau_i} \text{ bound in [29]}$$

$$\mathrm{C}_\Sigma + m'\,\mathrm{C}_k + (A_k + \mathrm{D}_k)U_\mathcal{T} + U > \frac{\Theta}{\Pi}(A_k + \mathrm{D}_k) - B \qquad \text{Equation (8.2)}$$

$$A_k < \frac{\mathrm{C}_\Sigma + m\,\mathrm{C}_k - \mathrm{D}_k\left(\frac{\Theta}{\Pi} - U_\mathcal{T}\right) + U + B}{\frac{\Theta}{\Pi} - U_\mathcal{T}}$$

□

It can also be observed that Equation (8.5) only needs to be evaluated at those values of A_k for which at least one of $\hat{I}_{i,2}, \bar{I}_{i,2}$, or sbf_μ change. Therefore, Theorem 8.1 gives a pseudo-polynomial time schedulability condition whenever utilization $U_\mathcal{T}$ is strictly less than resource bandwidth $\frac{\Theta}{\Pi}$. In our techniques described later, we compute minimum possible Θ and minimum required m' for a given value of Π. Since Θ appears inside floor and ceiling functions in sbf_μ, these computations may be intractable. Therefore, we replace sbf_μ in Theorem 8.1 with lsbf_μ from Equation (8.2), before using the theorem to generate MPR interfaces.

Discussions. We have only focused on one intra-cluster scheduling algorithm in this chapter. However, our analysis technique can be easily extended to other intra-cluster scheduling algorithms. Specifically, in the schedulability condition given in Equation (8.5), $\mathrm{DEM}(A_k + \mathrm{D}_k, m')$ depends on gEDF, and $\mathrm{sbf}_\mu(A_k + \mathrm{D}_k)$ depends on MPR model μ. Suppose there exists a function $\mathrm{DEM}_{\mathrm{DM}}(A_k + \mathrm{D}_k, m')$ that can compute workload upper bound for a task set scheduled under

global DM. Then, we can plug in $\mathrm{DEM}_{\mathrm{DM}}(A_k + \mathrm{D}_k, m')$ into Equation (8.5) to derive a schedulability condition for global DM intra-cluster scheduling. In fact, such a $\mathrm{DEM}_{\mathrm{DM}}$ can be obtained by extending current results over dedicated resource [33].

Bertogna and Cirinei have derived an upper bound for the worst-case response time of tasks scheduled under gEDF or global DM [31]. They have also used this bound to improve the carry-in demand CI_i that we use in our schedulability condition. However, this improvement to the carry-in demand cannot be applied in our case. Since we schedule tasks using MPR model, any response time computation depends on the resource supply as well, in addition to task demand. Then, to use the response time bounds presented in [31], we must extend it with sbf of MPR model. However, since we are computing the MPR model (capacity Θ and concurrency m'), its sbf is unknown and therefore the response time is not computable. One way to resolve this issue is to compute Θ and m' using binary search. However, since Θ belongs to the domain of non-negative real numbers, binary search for the minimum Θ can take a prohibitively long time.

8.4 Component interface generation

In this section, we develop a technique to generate MPR interface $\mu = \langle \Pi, \Theta, m' \rangle$, for a cluster $\mathcal{C}$ comprising of sporadic tasks $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ scheduled under gEDF. For this purpose, we use the schedulability condition given by Theorem 8.1. We assume that period Π of interface μ is specified a priori by system designer. For instance, one can specify this period taking into account preemption overheads in the system. We then compute values for capacity Θ and number of processors m' so that resource bandwidth of the interface is minimized. Finally, we also develop a technique that transforms MPR interfaces to periodic tasks, in order to schedule clusters on the multiprocessor platform (inter-cluster scheduling).

8.4.1 Minimum bandwidth interface

It is desirable to minimize the resource bandwidth of MPR model μ when generating interface for $\mathcal{C}$, because $\mathcal{C}$ then consumes minimum possible resource from the multiprocessor platform. We now give a lemma which states that resource bandwidth required to guarantee schedulability of task set $\mathcal{T}$, monotonically increases as number of processors in the cluster increases.

Lemma 8.3 *Consider interfaces* $\mu_1 = \langle \Pi_1, \Theta_1, m'_1 \rangle$ *and* $\mu_2 = \langle \Pi_2, \Theta_2, m'_2 \rangle$*, such that* $\Pi_1 = \Pi_2$ *and* $m'_2 = m'_1 + 1$*. Suppose these two interfaces guarantee schedulability of the same component* $\mathcal{C}$ *with their smallest possible resource bandwidth, respectively. Then,* μ_2 *has a higher resource bandwidth than* μ_1 *does,* i.e.*,* $\Theta_1 < \Theta_2$*.*

Proof We prove this lemma by contradiction. Consider $\mu'_2 = \langle \Pi_2, \Theta'_2, m'_2 \rangle$ such that $\Theta'_2 \leq \Theta_1$. Suppose μ'_2 guarantees schedulability of component $\mathcal{C}$ as per Theorem 8.1.

Let δ_d denote the difference in resource requirements of $\mathcal{C}$ on m'_1 and m'_2 processors for some interval length $A_k + D_k$, *i.e.*, difference in function DEM used in Theorem 8.1. Then,

$$\begin{aligned}
\delta_d =& \mathrm{DEM}(A_k + \mathrm{D}_k, m'_2) - \mathrm{DEM}(A_k + \mathrm{D}_k, m'_1) \\
=& \sum_{i:i \in L_{(m'_2-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) - \sum_{i:i \in L_{(m'_1-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + (m'_2 - m'_1)\,\mathrm{C}_k \\
=& \sum_{i:i \in L_{(m'_2-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) - \sum_{i:i \in L_{(m'_1-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + \mathrm{C}_k \\
>& 0\ . \qquad (8.6)
\end{aligned}$$

It is indicated by $\delta_d > 0$ that the same component has a greater upper bound on resource demand when it executes on more processors. Now, let δ_s denote the difference in the linear supply bound

function between μ_1 and μ'_2 for interval length $A_k + D_k$, *i.e.*,

$$\begin{aligned}
\delta_s =& \text{lsbf}_{\mu'_2}(A_k + \text{D}_k) - \text{lsbf}_{\mu_1}(A_k + \text{D}_k) \\
=& \frac{\Theta'_2}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta'_2}{m'_2}\right)\right) - \frac{\Theta_1}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_1}{m'_1}\right)\right) \\
\leq& \frac{\Theta_1}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_1}{m'_2}\right)\right) - \frac{\Theta_1}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_1}{m'_1}\right)\right) \\
\leq& \frac{2(\Theta_1)^2}{\Pi_1}\left(\frac{1}{m'_2} - \frac{1}{m'_1}\right) \\
=& -\frac{2(\Theta_1)^2}{m'_1 m'_2 \Pi_1} \\
<& 0 \ . \qquad (8.7)
\end{aligned}$$

It is indicated by $\delta_s < 0$ that MPR models provide less resource supply with more available processors, when values of period and capacity are fixed. Thus, $\delta_d > 0$ and $\delta_s < 0$, for all $A_k + D_k$. Since μ_1 guarantees schedulability of component $\mathcal{C}$ using the smallest possible resource bandwidth, $\text{DEM}(A_k + D_k, m'_1) = \text{lsbf}_{\mu_1}(A_k + D_k)$ for some $A_k + D_k$. Then, $\text{DEM}(A_k + D_k, m'_2) > \text{lsbf}_{\mu'_2}(A_k + D_k)$ for that $A_k + D_k$, and therefore μ'_2 does not guarantee schedulability of $\mathcal{C}$ according to Theorem 8.1. This contradicts the assumption $\Theta'_2 \leq \Theta_1$. □

Lemma 8.3 suggests that when we generate interface μ, we use the smallest number of processors to minimize resource bandwidth of μ. However, an arbitrarily small number for m', say $m' = 1$, may result in an *infeasible* μ. Recall that a MPR model $\mu = \langle \Pi, \Theta, m' \rangle$ is defined to be feasible if and only if $\Theta \leq m' \Pi$. Therefore, we find a feasible interface μ for $\mathcal{C}$ that: (1) guarantees schedulability of $\mathcal{C}$ based on Theorem 8.1, and (2) uses the smallest possible number of processors (m^*). We can find such m^* through search. Since bandwidth is monotonic with number of processors, a binary search can be performed to determine m^*. For this search to terminate, both a lower and upper bound on m^* should be known. $\lceil U_{\mathcal{T}} \rceil$ is clearly a lower bound on the number of processors necessary to schedule $\mathcal{C}$, where $U_{\mathcal{T}} = \sum_i \frac{\text{C}_i}{\text{T}_i}$. If the number of processors on the multiprocessor platform is known, then that number can be used as an upper bound for m^*.

Otherwise, the following lemma gives an upper bound for m^* as a function of parameters of tasks in $\mathcal{T}$.

Lemma 8.4 *If* $m' \geq \frac{\sum_{i=1}^{n} \mathrm{C}_i}{\min_{i=1,\ldots,n}\{\mathrm{D}_i - \mathrm{C}_i\}} + n$, *then feasible MPR model* $\mu = \langle \Pi, m'\,\Pi, m' \rangle$ *guarantees schedulability of* $\mathcal{C}$ *as per Theorem 8.1.*

Proof

$$m' \geq \frac{\sum_{i=1}^{n} \mathrm{C}_i}{\min_{i=1,\ldots,n}\{\mathrm{D}_i - \mathrm{C}_i\}} + n$$

(Since $\forall k, A_k \geq 0$ in Theorem 8.1)

$$\Rightarrow m' \geq \frac{\sum_{i=1}^{n} \mathrm{C}_i}{A_k + \mathrm{D}_k - \mathrm{C}_k} + n \qquad \forall k \text{ and } \forall A_k$$

$$\Rightarrow m'(A_k + \mathrm{D}_k - \mathrm{C}_k) \geq \sum_{i=1}^{n} \mathrm{C}_i + n(A_k + \mathrm{D}_k - \mathrm{C}_k) \qquad \forall k \text{ and } \forall A_k \tag{8.8}$$

Now consider the function $\mathrm{DEM}(A_k + \mathrm{D}_k, m')$ from Theorem 8.1.

$$\mathrm{DEM}(A_k + \mathrm{D}_k, m') = \sum_{i=1}^{n} \hat{I}_{i,2} + \sum_{i:i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + m'\,\mathrm{C}_k$$

$$\left(\text{Since each } \hat{I}_{i,2} \leq A_k + \mathrm{D}_k - \mathrm{C}_k \text{ and } \sum_{i:i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) \leq \sum_{i=1}^{n} \mathrm{C}_i \right)$$

$$\Rightarrow \mathrm{DEM}(A_k + \mathrm{D}_k, m') \leq n(A_k + \mathrm{D}_k - \mathrm{C}_k) + \sum_{i=1}^{n} \mathrm{C}_i + m'\,\mathrm{C}_k$$

(From Equation (8.8))

$$\Rightarrow \mathrm{DEM}(A_k + \mathrm{D}_k, m') \leq m'(A_k + \mathrm{D}_k - \mathrm{C}_k) + m'\,\mathrm{C}_k$$

$$= \mathrm{sbf}_{\mu}(A_k + \mathrm{D}_k)$$

Since this inequality holds for all k and A_k, from Theorem 8.1 we get that μ is guaranteed to schedule component $\mathcal{C}$. □

Since μ in Lemma 8.4 is feasible and guarantees schedulability of $\mathcal{C}$, $\frac{\sum_{i=1}^{n} \mathrm{C}_i}{\min_{i=1}^{n}\{\mathrm{D}_i - \mathrm{C}_i\}} + n$ is an upper bound for m^*. Thus, we generate an interface for $\mathcal{C}$ by doing a binary search for m^* in

the range $[\lceil U_{\mathcal{T}} \rceil, \frac{\sum_{i=1}^{n} \mathrm{C}_i}{\min_{i=1}^{n}\{\mathrm{D}_i - \mathrm{C}_i\}} + n]$. For each value of number of processors m', we compute the smallest value of Θ that satisfies Equation (8.5) in Theorem 8.1, assuming sbf_μ is replaced with lsbf_μ. Θ $(= \Theta^*)$ corresponding to the smallest value of m' that guarantees schedulability of $\mathcal{C}$ and results in a feasible interface, is then chosen as the capacity of μ. Also, m^* is chosen as the value for number of processors in the cluster, *i.e.*, $\mu = \langle \Pi, \Theta^*, m^* \rangle$.

Algorithm complexity. To bound A_k as in Theorem 8.2 we must know the value of Θ. However, since Θ is being computed, we use its smallest (0) and largest $(m' \Pi)$ possible values to bound A_k. For each value of $m' > U_{\mathcal{T}}$, Θ can then be computed in pseudo-polynomial time using Theorem (8.1), assuming sbf is replaced with lsbf. This follows from the fact that the denominator in the bound of A_k in Theorem 8.2 is non-zero. The only problem case is when $m' = \lceil U_{\mathcal{T}} \rceil = U_{\mathcal{T}}$. However, in this case we now show that $\mu = \langle \Pi, m' \Pi, m' \rangle$ can schedule $\mathcal{C}$ if and only if, $m' = 1$ and all tasks in $\mathcal{T}$ have implicit deadlines. Clearly, for constrained deadline task systems, a resource bandwidth of $U_{\mathcal{T}}$ is not sufficient to guarantee schedulability. Now suppose $m' > 1$. Then, the LHS of Equation (8.5) is $> \sum_{i=1}^{n} \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k) \geq (A_k + \mathrm{D}_k) U_{\mathcal{T}}$, because $\hat{I}_{i,2} = \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k)$, $\bar{I}_{i,2} \geq \mathrm{dbf}_{\tau_i}(A_k + \mathrm{D}_k)$, and $m' \mathrm{C}_k > 0$. Hence, in this case $m' > U_{\mathcal{T}}$, which is a contradiction. Therefore, computing the interface for $m' = U_{\mathcal{T}}$ can be done in constant time. The number of different values of m' to be considered is polynomial in input size, because the search interval is bounded by numbers that are polynomial in input parameters. Therefore, the entire interface generation process has pseudo-polynomial complexity.

Example 8.1 *Consider the example virtual clustering framework shown in Figure 8.2. Let clusters $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ be assigned tasks as shown in Table 8.1. Interfaces μ_1^*, μ_2^*, and μ_3^*, for clusters $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$, are shown in Figures 8.9(a), 8.9(b), and 8.9(c) respectively. In the figures, we have plotted the resource bandwidth of these interfaces for varying periods, and m' denotes number of processors in the cluster.*

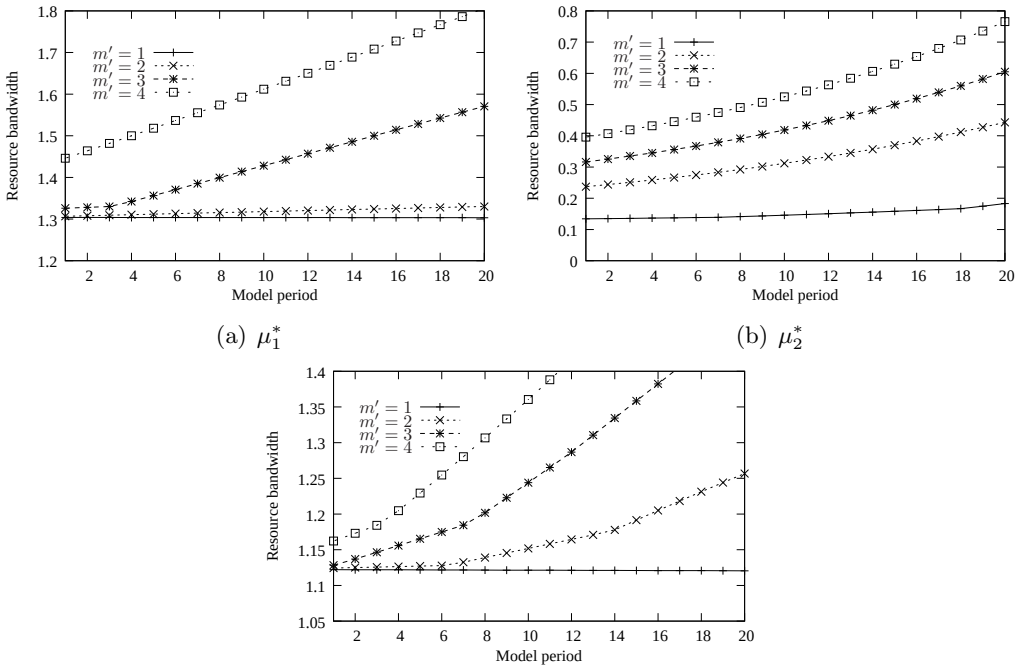
Cluster	Task set	$\sum_i \frac{C_i}{T_i}$	$\sum_i \frac{C_i}{D_i}$
$\mathcal{C}_1$	$\{(60,5,60),(60,5,60),(60,5,60),(60,5,60),(70,5,70),(70,5,70),$ $(80,5,80),(80,5,80),(80,10,80),(90,5,90),(90,10,90),$ $(90,10,90),(100,10,100),(100,10,100),(100,10,100)\}$	1.304	1.304
$\mathcal{C}_2$	$\{(60,5,60),(100,5,100)\}$	0.1333	0.1333
$\mathcal{C}_3$	$\{(45,2,40),(45,2,45),(45,3,40),(45,3,45),(50,5,45),$ $(50,5,50),(50,5,50),(50,5,50),(70,5,60),(70,5,60),$ $(70,5,65),(70,5,65),(70,5,65),(70,5,65),(70,5,70)\}$	1.1222	1.1930

Table 8.1: Clusters $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$

Figures 8.9(a) and 8.9(c) show that when $m' = 1$, interfaces μ_1^* and μ_3^* are not feasible; their resource bandwidths are greater than 1 for all period values. This shows that clusters $\mathcal{C}_1$ and $\mathcal{C}_3$ are not schedulable on clusters having one processor. This is as expected, because the utilization of task sets in these clusters is also greater than one. However, when $m' = 2$, μ_1^* and μ_3^* are feasible, *i.e.*, their respective resource bandwidths are at most two. Therefore, for clusters $\mathcal{C}_1$ and $\mathcal{C}_3$, we choose MPR interfaces μ_1^* and μ_3^* with $m' = 2$. Similarly, Figure 8.9(b) shows that μ_2^* is a feasible interface for cluster $\mathcal{C}_2$ when $m' = 1$. These plots also show that resource overheads[3] incurred by our interfaces are small for the non-trivial examples presented here.

Discussions. In hierarchical uniprocessor systems, EDP resource models described in Chapter 5 generalize periodic resource models and lead to better resource utilization. Analogously, for hierarchical multiprocessor systems, we can consider a generalization of MPR models called *multiprocessor explicit deadline periodic* (MEDP) resource models. A MEDP resource model $\langle \Pi, \Theta, \Delta, m' \rangle$ specifies that an identical unit-capacity multiprocessor platform collectively provides Θ units of resource in Δ units of time with this supply pattern repeating every Π time units. Again, the amount of concurrency in this processor supply is bounded by m'. It is possible to

[3]Difference between $\max_k \max_{A_k} \frac{\text{DEM}(A_k+D_k,m')}{A_k+D_k}$ and resource bandwidth.



(a) μ_1^* (b) μ_2^*

(c) μ_3^*

Figure 8.9: MPR model based interfaces

use MEDP resource models in our framework by replacing sbf_μ in Theorem 8.1 with the sbf of a MEDP model. Techniques developed in this section can be combined with those developed in Chapter 5 to compute the three unknown parameters Θ, Δ, and m'. In spite of the advantages of MEDP resource models, we focus on MPR models in this chapter. This is because the inter-cluster scheduling framework that we develop below (see Section 8.5.1), negates the advantages of MEDP models by imposing restrictions on the period of MPR models. Systems in which such restrictions are not desirable, can use MEDP models to improve their resource utilization.

8.4.2 Inter-cluster scheduling

As discussed in the introduction, virtual clustering involves two-level scheduling; scheduling of tasks within each cluster (intra-cluster scheduling), and scheduling of clusters on the multiprocessor platform (inter-cluster scheduling). MPR interfaces generated in the previous section capture task-level concurrency constraints within a cluster. Hence, inter-cluster scheduling need not worry

about these constraints when it schedules cluster interfaces. However, there is no known scheduling algorithm for MPR interfaces. Therefore, we now develop a technique to transform a MPR model into periodic tasks, such that resource requirements of these tasks are at least as much as those of the resource model.

Definition 8.2 *Consider a MPR model* $\mu = \langle \Pi, \Theta^*, m^* \rangle$*, and let* $\alpha = \Theta^* - m^* \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor$ *and* $k = \lfloor \alpha \rfloor$*. Define the transformation from* μ *to a periodic task set* $\mathcal{T}_\mu$ *as* $\mathcal{T}_\mu = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_{m^*} = (\mathrm{T}_{m^*}, \mathrm{C}_{m^*}, \mathrm{D}_{m^*})\}$*, where* $\tau_1 = \ldots = \tau_k = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor + 1, \Pi\right)$*,* $\tau_{k+1} = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor + \alpha - k \left\lfloor \frac{\alpha}{k} \right\rfloor, \Pi\right)$*, and* $\tau_{k+2} = \ldots = \tau_{m^*} = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor, \Pi\right)$*.*

In this definition, it is easy to see that the total resource demand of $\mathcal{T}_\mu$ is Θ^* in every period Π. Further, we have assumed that whenever Θ^* is not an integer, resource supply from μ fully utilizes one processor before moving to the next processor. For example, if $\Theta^* = 2.5$ and $m^* = 3$, then μ will provide two units of resource from two processors and the remaining 0.5 units from the third processor. The following theorem proves correctness of this transformation.

Theorem 8.5 *If all the deadlines of task set* $\mathcal{T}_\mu$ *in Definition 8.2 are met by some resource supply with concurrency at most* m^* *at any time instant, then its supply bound function is lower bounded by* sbf_μ*.*

Proof Since $\mathcal{T}_\mu$ has m^* tasks, it can utilize at most m^* processors at any time instant. Therefore, if some resource supply provides more than m^* processors at any time instant, then we can ignore these additional processor allocations. Hence, we only need to consider resource supplies with concurrency at most m^*.

Total resource demand of all the tasks in $\mathcal{T}_\mu$ is Θ^* in every period of Π time units. Then, to meet all the deadlines of task set $\mathcal{T}_\mu$, any resource supply must provide at least Θ^* resource units in every period of Π, with amount of concurrency at most m^*. But, this is exactly the definition

of MPR model $\mu = \langle \Pi, \Theta^*, m^* \rangle$. Therefore, the supply bound function of this resource supply is lower bounded by sbf_μ. □

Thus, using Definition 8.2, MPR interfaces generated in the previous section can be transformed into periodic tasks. Once such tasks are generated for each virtual cluster, inter-cluster scheduling can be done using existing multiprocessor algorithms like gEDF, Pfair [25], etc.

Example 8.2 *For MPR interfaces* μ_1^*, μ_2^*, *and* μ_3^* *generated in Example 8.1, we select period values* $6, 8$, *and* 5 *respectively. The corresponding MPR interfaces are* $\langle 6, 8.22, 2 \rangle, \langle 8, 2.34, 1 \rangle$, *and* $\langle 5, 5.83, 2 \rangle$. *Using Definition 8.2 we get periodic task sets* $\mathcal{T}_{\mu_1^*} = \{(6,5,6),(6,4,6)\}, \mathcal{T}_{\mu_2^*} = \{(8,3,8)\}$, *and* $\mathcal{T}_{\mu_3^*} = \{(5,3,5),(5,3,5)\}$. *Suppose the three clusters* $\mathcal{C}_1, \mathcal{C}_2$, *and* $\mathcal{C}_3$ *(*i.e.*, task set* $\{\mathcal{T}_{\mu_1^*}, \mathcal{T}_{\mu_2^*}, \mathcal{T}_{\mu_3^*}\}$*) are scheduled on a multiprocessor platform using g*EDF. *Then the resulting MPR interface* μ^* *is plotted in Figure 8.10.*

Figure 8.10 shows that when $m' = 3$, interface μ^* is not feasible; its resource bandwidth is greater than 3 for all period values. This shows that component $\mathcal{C}_4$ is not schedulable on clusters having 3 processors. However, from the figure, it can also be seen that $\mathcal{C}_4$ is schedulable on a multiprocessor platform having 4 processors (μ^* is feasible when $m' = 4$). This example also shows the advantage of virtual clustering over physical clustering. The three components $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$, would require 5 processors under physical clustering (2 for $\mathcal{C}_1$ and $\mathcal{C}_3$ each, and 1 for $\mathcal{C}_2$). On the other hand, a gEDF based virtual clustering technique can schedule these clusters using only 4 processors. Although total utilization of tasks in the three clusters is 2.56, our analysis requires 4 processors to schedule the system. This overhead is as a result of the following factors: (1) gEDF is not an optimal scheduling algorithm on multiprocessor platforms (both for intra- and inter-cluster scheduling), (2) the schedulability conditions we use are only sufficient conditions, and (3) capturing task-level concurrency constraints in a component interface leads to some increase in resource requirements (resource overhead of abstracting cluster into MPR interface).

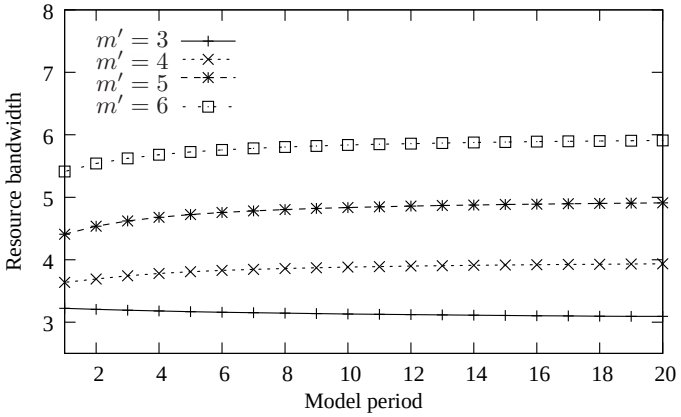


Figure 8.10: MPR interface μ^*

8.5 Virtual cluster-based scheduling algorithms

In this section we propose new virtual-cluster based scheduling algorithms for implicit deadline sporadic task systems. Prior to presenting these algorithms, we eliminate resource overheads from the virtual clustering framework proposed in Section 8.4.

8.5.1 Improved virtual-clustering framework

In this section we present an inter-cluster scheduling algorithm that is optimal whenever all the MPR interfaces being scheduled under it have identical periods; if the interfaces are schedulable under some scheduling algorithm, then our inter-cluster scheduler can also schedule it. We then present another transformation from MPR models to periodic tasks, which along with the optimal inter-cluster scheduler, results in an improved sbf for MPR models. These two together, eliminate the resource overheads described at the end of previous section.

McNaughton [86] presented an algorithm for scheduling real-time jobs in a given time interval on a multiprocessor platform. This algorithm can be explained as follows: Consider n jobs to be scheduled on m processors in a time interval $(t_1, t_2]$ of length t, such that no job is simultaneously scheduled on more than one processor. The job set need not be sorted in any particular order. McNaughton's algorithm schedules the i^{th} job on the first non-empty processor, packing jobs from

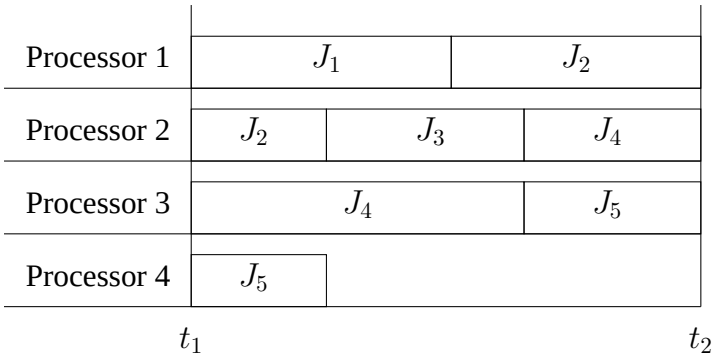


Figure 8.11: Schedule of job set under McNaughton's algorithm in interval $(t_1, t_2]$

left to right. Suppose $(i-1)^{st}$ job was scheduled on processor k up to time instant $t_3 (t_1 \leq t_3 \leq t_2)$. Then, up to $t_2 - t_3$ time units of the i^{th} job are scheduled on processor k, and the remaining time units are scheduled on processor $k+1$ starting from t_1. Figure 8.11 illustrates this schedule for a job set $\{J_1, \ldots, J_5\}$ on 4 processors. Note that if the total resource demand of a job is at most $t_2 - t_1$, then (1) the job is scheduled on at most two processors by McNaughton's algorithm, and (2) the job is never scheduled simultaneously on both the processors. The following theorem establishes conditions under which this algorithm can successfully schedule job sets.

Theorem 8.6 (Theorem 3.1 in [86]) *Let* $\mathrm{c}_1, \ldots, \mathrm{c}_n$ *denote number of time units of the* n *jobs to be scheduled in the interval* $(t_1, t_2]$ *on* m *identical, unit-capacity processors. If* $\sum_{i=1}^{n} \mathrm{c}_i \leq m(t_2 - t_1)$*, then a necessary and sufficient condition to guarantee schedulability of this job set is that for all* i*,* $\mathrm{c}_i \leq t_2 - t_1$*.*

If $\mathrm{c}_i > t_2 - t_1$, then the i^{th} job cannot be scheduled in interval $(t_1, t_2]$ by any scheduling algorithm, unless the job is simultaneously scheduled on more than one processor. Likewise, if $\sum_{i=1}^{n} \mathrm{c}_i > m(t_2 - t_1)$, then also the job set cannot be scheduled by any scheduling algorithm, because total resource demand in the interval $(t_1, t_2]$ is greater than total available resource. Hence, Theorem 8.6 in fact shows that McNaughton's algorithm is optimal for scheduling job sets in a given time interval.

Consider a periodic task set $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$, where $\mathrm{T}_1 =$

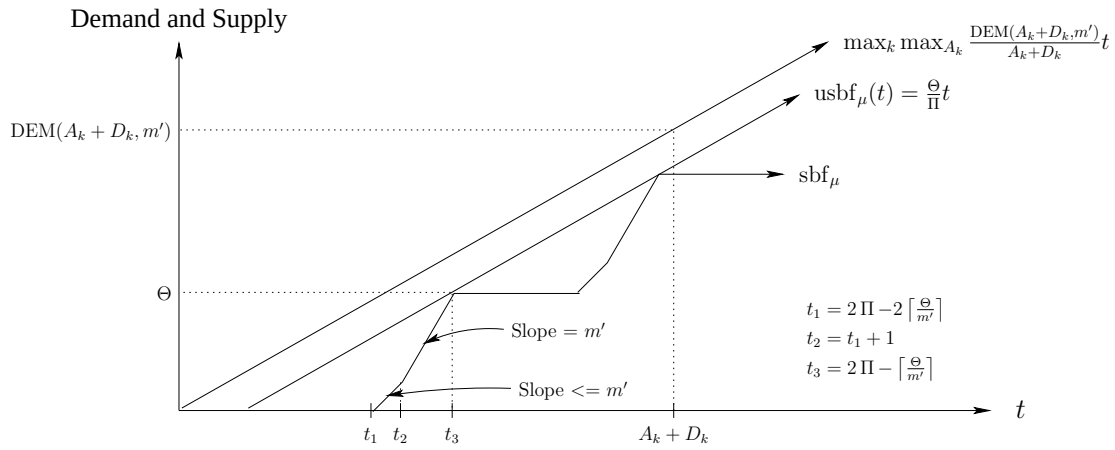


Figure 8.12: sbf_μ and its linear upper bound usbf_μ

$\ldots = \mathrm{T}_n = \mathrm{D}_1 = \ldots = \mathrm{D}_n\ (= \mathrm{T})$. Tasks in $\mathcal{T}$ have identical periods and implicit deadline. Suppose we use McNaughton's algorithm in intervals $(k\,\mathrm{T}, (k+1)\,\mathrm{T}]$, $k \in \mathbb{I}$, to schedule jobs of $\mathcal{T}$ on m identical, unit-capacity processors. Then, for each interval $(k\,\mathrm{T}, (k+1)\,\mathrm{T}]$, (1) all jobs of $\mathcal{T}$ are released at the beginning of the interval $(k\,\mathrm{T})$, and (2) all jobs of $\mathcal{T}$ have deadline at the end of the interval $((k+1)\,\mathrm{T})$. Therefore, from Theorem 8.6 we get that McNaughton's algorithm optimally schedules these jobs in each interval, and this leads to the following direct corollary.

Corollary 8.7 *Let* $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\}$ *denote a periodic task set to be scheduled on* m *identical, unit-capacity processors. If* $\mathrm{T}_1 = \ldots = \mathrm{T}_n = \mathrm{D}_1 = \ldots = \mathrm{D}_n\ (= \mathrm{T})$, *then a necessary and sufficient condition for* $\mathcal{T}$ *to be schedulable using McNaughton's algorithm is that* $\sum_{i=1}^{n} \mathrm{C}_i \leq m\,\mathrm{T}$ *and* $\mathrm{C}_i \leq \mathrm{T}_i$ *for each* i.

Consider the virtual clustering framework proposed in Section 8.4. Suppose all the MPR interfaces in this framework have identical periods. Then, all the periodic tasks generated using Definition 8.2 also have identical period values. From Corollary 8.7 we get that McNaughton's algorithm is optimal for scheduling these tasks on the physical platform, *i.e.*, the algorithm does not incur any resource overhead for inter-cluster scheduling.

Another source of resource overhead is the abstraction of a cluster into MPR interface and its

transformation to a periodic task set. This overhead results from the sub-optimality of sbf of MPR models which can be explained as follows. Consider the two functions, sbf_μ and usbf_μ, shown in Figure 8.12. The resource bandwidth used by MPR model μ is equal to the slope of line usbf_μ. Suppose μ is used to abstract the resource demand of cluster $\mathcal{C}$ in Theorem 8.1. Then, since sbf_μ has a non-zero x-axis intercept, the bandwidth of μ is strictly larger than the schedulability load, $\max_k \max_{A_k} \text{DEM}(A_k + \text{D}_k, m')/(A_k + \text{D}_k)$, of cluster $\mathcal{C}$. If not, then as shown in Figure 8.12, there exists some $A_k + D_k$ for which Theorem 8.1 is not satisfied. This explains the resource overhead in abstraction of clusters to MPR interfaces. Now, suppose μ is transformed into the periodic task set $\mathcal{T}_\mu$ using Definition 8.2. Then, from Theorem 8.5 we get that the total resource demand of $\mathcal{T}_\mu$ is at least as much as sbf_μ. However, since sbf_μ does not guarantee Θ^* resource units in an interval of length Π (see Figure 8.12), a resource supply with supply bound function exactly sbf_μ cannot schedule $\mathcal{T}_\mu$. This explains the resource overhead in transformation of MPR interfaces to periodic tasks.

To eliminate the aforementioned overheads, we must modify the transformation from MPR interface μ to periodic task set $\mathcal{T}_\mu$ presented in Definition 8.2. This is because the schedule of $\mathcal{T}_\mu$ determines the resource supply from the multiprocessor platform to interface μ, and this in turn determines sbf_μ. We now present a new transformation from MPR models to periodic tasks as follows.

Definition 8.3 *Given a MPR model $\mu = \langle \Pi, \Theta^*, m^* \rangle$, we define its transformation to a periodic task set $\mathcal{T}_\mu$ as $\mathcal{T}_\mu = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_{m^*} = (\text{T}_{m^*}, \text{C}_{m^*}, \text{D}_{m^*})\}$, where $\tau_1 = \ldots = \tau_{m^*-1} = (\Pi, \Pi, \Pi)$ and $\tau_{m^*} = (\Pi, \Theta^* - (m^* - 1)\,\Pi, \Pi)$.*

In this definition, it is easy to see that the total resource demand of $\mathcal{T}_\mu$ is Θ^* in every Π time units, with concurrency at most m^*. Therefore, Theorem 8.5 holds in this case as well, *i.e.*, if all the deadlines of task set $\mathcal{T}_\mu$ are met by some resource supply with concurrency at most m^* at

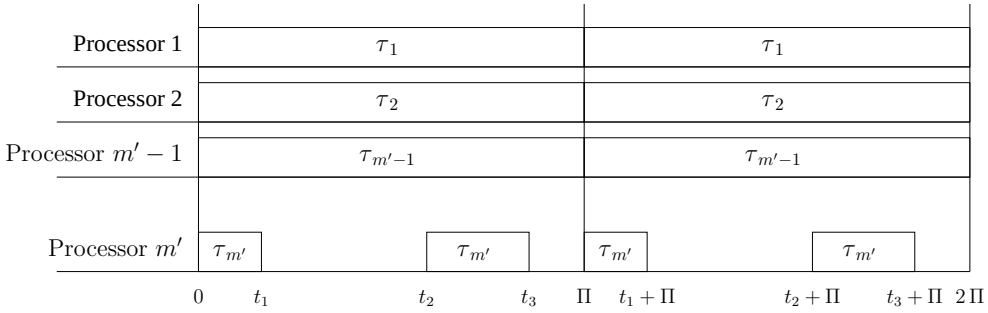


Figure 8.13: McNaughton's schedule of implicit deadline periodic tasks with same periods

any time instant, then its supply bound function is lower bounded by sbf_μ.

Now, suppose a cluster is abstracted into MPR interface $\mu = \langle \Pi, \Theta, m' \rangle$, which is then transformed into task set $\mathcal{T}_\mu$ using Definition 8.3. Let $\mathcal{T}_\mu$ be scheduled on the multiprocessor platform using McNaughton's algorithm, along with periodic tasks that all have period and deadline Π (implicit deadline task system with identical periods). Figure 8.13 illustrates the McNaughton schedule for task set $\mathcal{T}_\mu$. As can be seen in the figure, tasks $\tau_1, \ldots, \tau_{m'-1}$ completely utilize $m'-1$ processors on the platform. Furthermore, every job of task $\tau_{m'}$ is scheduled in an identical manner within its execution window (intervals $(0, t_1]$ and $(t_2, t_3]$ relative to release time). Since this schedule of $\mathcal{T}_\mu$ is used as resource supply for the underlying MPR interface, μ guarantees Θ resource units in any time interval of length Π, $2\,\Theta$ resource units in any time interval of length $2\,\Pi$, and so on. In other words, the blackout interval of sbf_μ described in Section 8.2.1 reduces to zero. The resulting sbf is plotted in Figure 8.14, and it is given by the following equation.

$$\text{sbf}_\mu(t) = \left\lfloor \frac{t}{\Pi} \right\rfloor \Theta + (t - \left\lfloor \frac{t}{\Pi} \right\rfloor \Pi) m' - \min\{t - \left\lfloor \frac{t}{\Pi} \right\rfloor \Pi, m'\,\Pi - \Theta\} \tag{8.9}$$

sbf_μ guarantees Θ resource units in any time interval of length Π. Then, a resource supply with supply bound function equal to sbf_μ can successfully schedule task set $\mathcal{T}_\mu$. Thus, we have eliminated the resource overhead that was present in the previous transformation given in Definition 8.2.

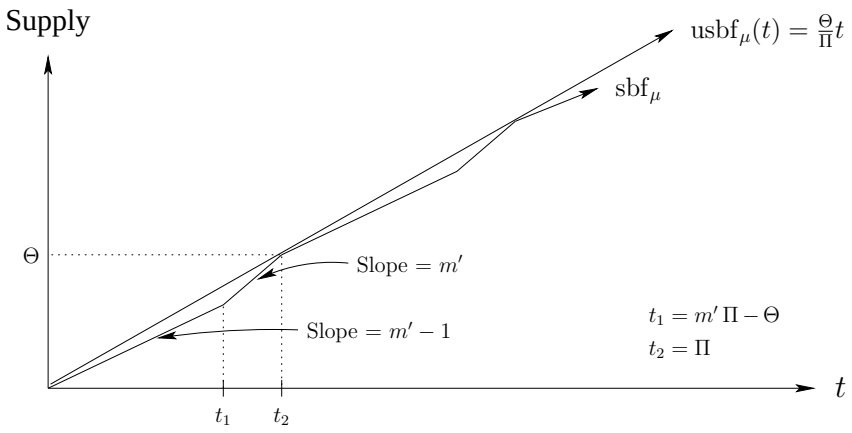


Figure 8.14: Improved sbf_μ and its linear upper bound usbf_μ

Now consider the schedulability condition for cluster $\mathcal{C}$ given by Equation (8.5) in Theorem 8.1. This equation needs to be evaluated for all values of A_k up to the bound given in Theorem 8.2 and for all tasks τ_k in cluster $\mathcal{C}$. In this equation, it is easy to see that $\text{DEM}(A_k + \text{D}_k, m')$ increases by at most $m' - 1$ for every unit increase in A_k, as long as $A_k + 1 + \text{D}_k$ does not coincide with the release or deadline of some task in cluster $\mathcal{C}$. In other words, $\text{DEM}(A_k + 1 + \text{D}_k, m') \leq \text{DEM}(A_k + \text{D}_k, m') + m' - 1$, whenever $A_k + 1 + \text{D}_k$ is not equal to $l\,\text{T}_i$ or $l\,\text{T}_i + \text{D}_i$ for any l and i (denoted as property *bounded increase*). This is because over such unit increases in A_k, $m'\,\text{C}_k$ and each $\hat{I}_{i,2}$ remain constant, and $\sum_{i:i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2})$ increases by at most $m' - 1$. However, sbf_μ increases by at least $m' - 1$ over each unit time interval (see Figure 8.14). Therefore, to generate interface μ, it is sufficient to evaluate Equation (8.5) at only those values of A_k for which $A_k + \text{D}_k$ is equal to $l\,\text{T}_i$ or $l\,\text{T}_i + \text{D}_i$, for some l and i. Now, suppose period Π of interface μ is equal to the GCD (greatest common divisor) of periods and deadlines of all tasks in cluster $\mathcal{C}$. Then, all required evaluations of Equation (8.5) will occur at time instants t for which $\text{sbf}_\mu(t) = \text{usbf}_\mu(t) = \frac{\Theta}{\Pi}t$ (see Figure 8.14). In other words, the RHS of Equation (8.5) can be replaced with $\frac{\Theta}{\Pi}t$. This means that the resource bandwidth of resulting interface μ ($\frac{\Theta}{\Pi}$) will be equal to the schedulability load, $\max_k \max_{A_k} \frac{\text{DEM}(A_k + D_k, m')}{A_k + D_k}$, of cluster $\mathcal{C}$. Thus, we have eliminated the resource overhead that was previously present in the cluster abstraction process.

We now summarize the contributions of this section. The following theorem, which is a direct consequence of above discussions, states the fundamental result of this section. This theorem states that our improved virtual-clustering framework does not incur any resource overheads in transforming MPR interfaces to periodic tasks, or in scheduling the transformed tasks on the multiprocessor platform.

Theorem 8.8 *Consider interfaces* $\mu_1 = \langle \Pi, \Theta_1, m'_1 \rangle, \ldots, \mu_p = \langle \Pi, \Theta_n, m'_n \rangle$. *Suppose they are transformed to periodic tasks using Definition 8.3. McNaughton's algorithm can successfully schedule the transformed tasks on* m *identical, unit-capacity multiprocessors if and only if,* $\sum_{i=1}^{p} \frac{\Theta_i}{\Pi} \leq m$.

Suppose (1) we want to schedule a constrained deadline sporadic task set $\mathcal{T}$ using virtual clusters on m identical, unit-capacity processors, (2) task-cluster mapping is given, and (3) each intra-cluster scheduler is such that the corresponding schedulability condition satisfies *bounded increase* property described above (*e.g.*, gEDF). Let (1) each virtual cluster be abstracted into an MPR interface whose period Π is equal to the GCD of periods and deadlines of all tasks in $\mathcal{T}$, (2) these interfaces be transformed into periodic tasks using Definition 8.3, and (3) these periodic tasks be scheduled on the multiprocessor platform using McNaughton's algorithm. Then, in addition to the results stated in Theorem 8.8, the resource bandwidth of each MPR interface will be equal to the schedulability load of corresponding cluster.

8.5.2 Virtual clustering of implicit deadline task systems

In this section we propose two virtual cluster-based scheduling algorithms for implicit deadline sporadic task sets. We consider the problem of scheduling an implicit deadline sporadic task set $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{T}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{T}_n)\}$ on m identical, unit-capacity processors. We first present a new virtual cluster-based scheduler that is optimal like the well known Pfair algorithm [104], but unlike Pfair, has a non-trivial bound on the number of preemptions. The second

technique extends the well known algorithm US-EDF$\{m/(2m-1)\}$ [105] with virtual clusters. We show that the presently known resource utilization bound of US-EDF$\{m/(2m-1)\}$ can be improved through this virtualization.

VC-IDT scheduling algorithm

In VC-IDT (*Virtual Clustering - Implicit Deadline Tasks*) scheduling algorithm we consider a trivial task-cluster mapping that assigns each task $\tau_i \in \mathcal{T}$ to its own virtual cluster $\mathcal{C}_i$ having one processor. Since each cluster has only one processor, we assume that each cluster uses EDF for intra-cluster scheduling[4]. Each virtual cluster $\mathcal{C}_i$ is abstracted into a MPR interface $\mu_i = \langle \Pi, \Theta_i, 1 \rangle$, where Π is equal to the GCD of $\mathrm{T}_1, \ldots, \mathrm{T}_n$ and $\frac{\Theta_i}{\Pi} = \frac{\mathrm{C}_i}{\mathrm{T}_i}$. Furthermore, each interface μ_i is transformed into periodic tasks using Definition 8.3, and the resulting task set is scheduled on the multiprocessor platform using McNaughton's algorithm. The following theorem proves that VC-IDT is an optimal algorithm for scheduling implicit deadline sporadic task systems on identical, unit-capacity multiprocessor platforms.

Theorem 8.9 *Consider sporadic tasks $\mathcal{T} = \{\tau_1 = (\mathrm{T}_1, \mathrm{C}_1, \mathrm{T}_1), \ldots, \tau_n = (\mathrm{T}_n, \mathrm{C}_n, \mathrm{T}_n)\}$. A necessary and sufficient condition to guarantee that $\mathcal{T}$ is schedulable on m identical, unit-capacity processors using VC-IDT algorithm is*

$$\sum_{i=1}^{n} \frac{\mathrm{C}_i}{\mathrm{T}_i} \leq m \tag{8.10}$$

Proof In VC-IDT, each virtual cluster $\mathcal{C}_i$ comprising of task τ_i, is abstracted to the MPR interface $\mu_i = \langle \Pi, \Theta_i, 1 \rangle$, where Π is equal to the GCD of $\mathrm{T}_1, \ldots, \mathrm{T}_n$ and $\frac{\Theta_i}{\Pi} = \frac{\mathrm{C}_i}{\mathrm{T}_i}$. The interface set $\mu_1, \ldots, \mu_n$, all having identical periods, are then transformed to periodic tasks using Definition 8.3 and scheduled on the platform using McNaughton's algorithm. Therefore, from Theorem 8.8 we

[4]Since each cluster also has only one task, any work conserving algorithm can be used for intra-cluster scheduling.

get that this interface set is schedulable on the multiprocessor platform if and only if, $\sum_{i=1}^{n} \frac{\Theta_i}{\Pi} \leq m$, *i.e.*, $\sum_{i=1}^{n} \frac{\mathrm{C}_i}{\mathrm{T}_i} \leq m$.

Thus, to prove this theorem, we only need to show that for each i, interface μ_i can schedule cluster $\mathcal{C}_i$. $\mathcal{C}_i$ comprises of sporadic task τ_i and uses EDF scheduler. Therefore, any resource supply that can guarantee C_i resource units in all time intervals of length T_i, can be used to schedule τ_i. But, from the sbf of model μ_i (Equation (8.9)), it is easy to see that μ_i guarantees C_i resource units in any time interval of length T_i. This proves the theorem. □

Equation (8.10) is known to be a necessary and sufficient feasibility condition for scheduling implicit deadline sporadic task systems on m identical, unit-capacity processors [25, 104]. Hence, VC-IDT is an optimal scheduling algorithm for this problem domain. The only other known optimal scheduler for this problem, to the best of our knowledge, is the PD^2 Pfair/ERfair algorithm [104]. Therefore, VC-IDT is the first optimal algorithm for sporadic tasks that is neither Pfair [25] nor ERfair [5]. Since P-fairness and ER-fairness are stricter requirements than deadline satisfaction, these algorithms are known to incur high number of preemptions in order to guarantee these fairness properties; they can potentially incur m preemptions in every time unit. In contrast, the number of preemptions incurred by VC-IDT has a non-trivial upper bound which can be explained as follows. When interfaces $\mu_1, \ldots, \mu_n$ are scheduled using McNaughton's algorithm, there are at most $m-1$ preemptions in every period of Π time units. Each preemption of an interface μ_i may result in a preemption in the execution of task τ_i in cluster $\mathcal{C}_i$. However, τ_i cannot incur any other preemptions, because $\mathcal{C}_i$'s workload is comprised of only task τ_i. Therefore, the entire sporadic task set will incur at most $m-1$ preemptions in every Π time units.

Virtual clustering for USEDF$\{m/(2m-1)\}$

US-EDF$\{m/(2m-1)\}$, proposed by Srinivasan and Baruah [105], is a global scheduling algorithm for implicit deadline sporadic task systems. Under this algorithm, each task with utilization $(\frac{\mathrm{C}}{\mathrm{T}})$

greater than $\frac{m}{2m-1}$ is given the highest priority, and the remaining tasks are scheduled based on gEDF. Past research [105] has shown that this algorithm has a resource utilization bound of $\frac{m^2}{2m-1}$, *i.e.*, any implicit deadline sporadic task set with total task utilization ($\sum_i \frac{\mathrm{C}_i}{\mathrm{T}_i}$) at most $\frac{m^2}{2m-1}$ can be scheduled by US-EDF$\{m/(2m-1)\}$ on m identical, unit-capacity processors.

Now, consider the following virtual cluster-based US-EDF$\{m/(2m-1)\}$ scheduling algorithm. Let each task with utilization greater than $\frac{m}{2m-1}$ be assigned to its own virtual cluster having one processor and using EDF (denoted as *high utilization cluster*), and all the remaining tasks be assigned to a single cluster using gEDF (denoted as *low utilization cluster*). Each cluster is abstracted to a MPR interface, such that period Π of each interface is equal to the GCD of $\mathrm{T}_1, \ldots, \mathrm{T}_n$. Each high utilization cluster is abstracted to interface $\langle \Pi, \Theta, 1 \rangle$, where $\frac{\Theta}{\Pi}$ is equal to utilization of task in the cluster (Theorem 8.9 proves correctness of this abstraction). The low utilization cluster is abstracted to interface $\mu_{low} = \langle \Pi, \Theta', m' \rangle$, where Θ' and m' are generated using techniques in Sections 8.4 and 8.5.1. Finally, these interfaces are transformed to periodic tasks using Definition 8.3, and the resulting task set is scheduled on the multiprocessor platform using McNaughton's algorithm.

We now derive a resource utilization bound for the virtual cluster-based US-EDF$\{m/(2m-1)\}$ algorithm described above. Suppose α denotes the total utilization of all high utilization tasks, *i.e.*, the total resource bandwidth of all MPR interfaces that represent high utilization clusters is α. Then, since all the interfaces that we generate have identical periods, from Theorem 8.8 we get that the maximum resource bandwidth available for μ_{low} is $m - \alpha$. This means that $\frac{\Theta'}{\Pi} \leq m - \alpha$ and $\alpha \leq m$ are necessary and sufficient conditions to guarantee schedulability of task set $\mathcal{T}$ under virtual cluster-based US-EDF$\{m/(2m-1)\}$.

Suppose $\alpha > m - 1$. Then, $m - \alpha < 1$ and $m' \leq 1$. The last inequality can be explained as follows. $m' = \left\lceil \frac{\Theta'}{\Pi} \right\rceil$, because m' is the smallest number of processors upon which the low utilization cluster is schedulable. Then, $\frac{\Theta'}{\Pi} \leq m - \alpha < 1$ implies $m' \leq 1$. In this case, the low

utilization cluster is scheduled on an uniprocessor platform, and gEDF reduces to EDF, an optimal uniprocessor scheduler with resource utilization bound $m - \alpha$. Therefore, virtual cluster-based US-EDF$\{m/(2m-1)\}$ is optimal whenever $\alpha > m - 1$, *i.e.*, it can schedule $\mathcal{T}$ if $\sum_{i=1}^{n} \frac{\mathrm{C}_i}{\mathrm{T}_i} \leq m$.

Now suppose $\alpha \leq m - 1$. To derive the resource utilization bound in this case, we use a utilization bound of gEDF that was developed by Goossens *et. al.* [62]. As per this bound, μ_{low} can support a low utilization cluster whose total task utilization is upper bounded by $\left(\frac{\Theta'}{\Pi} - \left(\frac{\Theta'}{\Pi} - 1\right) U_{max}\right)$, where U_{max} is the maximum utilization of any task in the cluster. Then, in this case, the resource utilization bound of virtual cluster-based US-EDF$\{m/(2m-1)\}$ is

$$\begin{aligned} &\alpha + \left(\frac{\Theta'}{\Pi} - \left(\frac{\Theta'}{\Pi} - 1\right) U_{max}\right) \\ &= \alpha + (m - \alpha - (m - \alpha - 1)\, U_{max}) \end{aligned}$$

Since $m - \alpha \geq 1$, the bound in the above equation is minimized when U_{max} is maximized. Substituting $U_{max} = \frac{m}{2m-1}$ (largest utilization of any task in the low utilization cluster), we get a resource utilization bound of

$$\begin{aligned} &\alpha + \left(m - \alpha\left(1 - \frac{m}{2m-1}\right) + \frac{m}{2m-1}\right) \\ &= \frac{\alpha(2m-1) + (m-\alpha)(m-1) + m}{2m-1} \\ &\geq \frac{\alpha(2m-1) + (m-\alpha)(m-1) + m}{2m-1} \\ &= \frac{m^2 + \alpha m}{2m-1} \end{aligned}$$

Thus, the resource utilization bound of virtual cluster-based US-EDF$\{m/(2m-1)\}$ is $\min\{m, \frac{m^2+\alpha m}{2m-1}\}$. It is easy to see that whenever $\alpha > 0$, this bound is greater than the presently known resource utilization bound of $\frac{m^2}{2m-1}$ for US-EDF$\{m/(2m-1)\}$.

8.6 Conclusions

In this chapter we have considered the idea of cluster-based scheduling on multiprocessor platforms, as an alternative to existing partitioned and global scheduling strategies. Cluster-based scheduling can be viewed as a two-level scheduling strategy. Tasks in a cluster are globally scheduled within the cluster (intra-cluster scheduling), and clusters are then scheduled on the multiprocessor platform (inter-cluster scheduling). We have further classified clustering into physical (one-to-one) and virtual (many-to-many), depending on the mapping between clusters and processors on the platform. Virtual clustering is more general and less sensitive to task-processor mappings than physical clustering.

Towards supporting virtual cluster-based scheduling, we have developed techniques for hierarchical scheduling in this chapter. Resource requirements and concurrency constraints of tasks within each cluster are first abstracted into MPR interfaces. These interfaces are then transformed into periodic tasks which are used for inter-cluster scheduling. We have also developed an efficient technique to minimize total processor utilization of individual clusters under gEDF. Finally, we showed that resource utilization bound of US-EDF$\{m/(2m-1)\}$ can be improved through virtualization, and also developed a new optimal scheduling algorithm for implicit deadline tasks.

We only focused on gEDF for intra-cluster and McNaughton's for inter-cluster scheduling. However, our approach of isolating the inter-cluster scheduler from task-level concurrency constraints is general, and can be adopted to other scheduling algorithms as well. Moreover, this generality also means that our technique enables clusters with different intra-cluster schedulers to be scheduled on the same platform. It would be interesting to generalize this framework by including other intra and inter-cluster scheduling algorithms, with an aim to solve some open problems in multiprocessor scheduling.

Chapter 9

Conclusions and Future work

9.1 Conclusions

In this dissertation, we addressed problems in two research areas; compositional schedulability analysis of hierarchical systems, and scheduling algorithms and analysis techniques for multiprocessor platforms. For hierarchical systems, we developed new compositional techniques that focus on two fundamental issues; reuse of component interfaces for efficient on-the-fly analysis, and minimization of resource overhead incurred by these interfaces. We developed incremental analysis techniques based on the periodic resource model, taking into consideration preemption overheads incurred by components (Chapter 4). This analysis is independent of the order in which components are considered, and therefore allows interfaces to be reused for change-analysis. It results in an efficient technique for various on-the-fly modifications such as dynamic voltage scaling of processors in response to changes in workloads, on-line admission tests for new workloads, etc.

In Chapter 5 we introduced a new resource model called explicit deadline periodic (EDP) resource model. It generalizes the periodic resource model with an additional deadline parameter. We showed that this generalization provides a tighter control over resource supply, and hence leads

to component interfaces with smaller resource overhead. We also developed efficient transformations from EDP resource models to periodic tasks, thus reducing the overall system-level resource overhead in comparison to periodic model based interfaces. Towards a better understanding of resource usage in hierarchical systems, we defined two notions of optimal resource utilization in Chapter 6. We showed that load optimality, which requires a component interface to have the same average resource usage as the component, is achievable. We also showed that, in general, existing resource model based techniques cannot achieve load optimality. On the other hand, for demand optimality, which requires a component interface to have the same resource demand as that of the component, we showed with examples that the interface generation process is exponentially hard. Finally, using data sets from avionics systems, we showed that the techniques developed in this dissertation result in efficient interfaces in practice.

In the latter part of this dissertation, we proposed a novel scheduling technique for identical, unit-capacity multiprocessor platforms. We virtualized the task-processor allocation problem using a two-level scheduling hierarchy. Tasks are first allocated to a set of virtual processor clusters, and the clusters themselves are then scheduled on the multiprocessor platform. To support such a framework, we abstracted each virtual cluster using a multiprocessor periodic resource (MPR) model. A MPR model generalizes the periodic resource model with an additional concurrency parameter. This parameter identifies the maximum number of physical processors that can used to schedule the cluster at any time instant. We developed techniques to efficiently abstract cluster workloads into MPR models, and also to optimally schedule those models on the physical platform. Finally, we used this technique to develop a new optimal scheduling algorithm for implicit deadline sporadic tasks, and also improved the resource utilization bound of US-EDF$\{m/(2m-1)\}$ scheduler.

9.2 Future research directions

Hierarchical scheduling. An important area for future research is characterizing resource usage in hierarchical systems, when analysis techniques are required to be incremental. This involves defining notions of optimal resource utilization under the restriction that only associative functions can be used to abstract component workloads. Further, efficient analysis techniques that achieve these notions of optimality need to be developed. Such techniques are essential for the successful application of hierarchical systems in resource-constrained embedded devices. These devices typically operate in highly dynamic environments, and therefore need support for efficient on-the-fly analysis.

Another important research area is development of analysis techniques that take into account resource overheads arising from various factors like preemptions. Preemption overheads significantly impact the resource requirements in embedded devices that are already resource-constrained. Furthermore, in hierarchical systems, since preemptions occur at each level of the scheduling hierarchy, preemptions have a larger impact on the overall resource usage than in elementary real-time systems. It is therefore essential to develop techniques that accurately count the number of preemptions in hierarchical systems.

Multiprocessor scheduling. Splitting a given task set into processor clusters (virtual or physical) is a fundamental problem that must be addressed if cluster-based scheduling has to succeed. Also, since virtual clustering allows separation of intra- and inter-cluster scheduling, various task-processor clustering techniques can be combined with different intra- and inter-cluster schedulers. It is an interesting research area to develop new clustering algorithms, as well as intra- and inter-cluster scheduling algorithms, that result in high resource utilization. Ideally, it would be good to have a family of virtualization based schedulers that inherit many of the advantages of partitioned scheduling on one hand (low preemption and migration overheads), and global

scheduling on the other (high resource utilization), without suffering from most of their pitfalls. Additionally, these algorithms should solve some of the important open problems in this domain; for instance, optimal scheduling of arbitrary deadline periodic task systems.

Another direction for future research is the consideration of new task/job parameters in priority-driven multiprocessor scheduling. One such parameter is the remaining execution time of jobs. Intuitively, concurrency in a schedule may be maximized by reducing the disparity in remaining execution time of active jobs, subject to satisfaction of job deadlines. Such priority assignment techniques could improve resource utilization on multiprocessor platforms, and thereby address some of the open problems.

Appendix A

Proof of correctness of algorithm EDF-COMPACT

Here we prove correctness of algorithm EDF-COMPACT that we presented in Section 4.4. For this purpose, we consider an elementary component $\mathcal{C} = \langle \mathcal{T}, \text{EDF} \rangle$, where $\mathcal{T} = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_n = (\text{T}_n, \text{C}_n, \text{D}_n)\}$ denotes a set of sporadic tasks. We show that interface $\mathcal{CI}_\mathcal{C}$ generated by EDF-COMPACT consists of minimum bandwidth resource models $\phi_k = \langle k, \Theta_k \rangle$, where k ranges from 1 to Π^*. In other words, we show that $\mathcal{I}_\mathcal{C} = \bigcup_{k=1}^{\Pi^*} \phi_k$. For each k, we assume that lsbf_{ϕ_k} intersects $\text{dbf}_\mathcal{C}$ at time instants t_k and t'_k, such that t_k denotes the smallest and t'_k denotes the largest such time instant.

Consider capacity Θ_1 generated in Line 1 of algorithm EDF-COMPACT. If $\Theta_1 = 1$, then the minimum resource bandwidth required to schedule component $\mathcal{C}$ is one. In this case lsbf_{ϕ_k} for all k are identical (slope one and blackout interval length zero), *i.e.*, $t_k = t_1$ for all k. EDF-COMPACT then correctly computes interface $\mathcal{I}_\mathcal{C}$ because t_1 is a significant time instant. In the remainder of this appendix we assume that $\frac{\Theta_k}{k} < 1$ for all k. We now state and prove several lemmas in order to prove correctness of EDF-COMPACT. We first show that the intersection

of lsbf_{ϕ_k} with time axis (*i.e.*, blackout interval $2(k-\Theta_k)$) and its resource bandwidth (*i.e.*, $\frac{\Theta_k}{k}$), both increase as period k increases.

Lemma A.1 *For each k, $1 \leq k < \Pi^*$, $2(k+1-\Theta_{k+1}) > 2(k-\Theta_k)$ and $\frac{\Theta_{k+1}}{k+1} > \frac{\Theta_k}{k}$.*

Proof Let $\phi'_{k+1} = \langle k+1, \Theta'_{k+1} \rangle$ represent a periodic resource model such that $2(k+1-\Theta'_{k+1}) = 2(k-\Theta_k)$. Then

$$
\begin{aligned}
& (k+1-k) > \frac{(k+1-k)\,\Theta_k}{k} && \text{Since } 1 > \frac{\Theta_k}{k} \\
\Rightarrow & \frac{(k+1-k+\Theta_k)}{k+1} > \frac{\Theta_k}{k} \\
\Rightarrow & \frac{\Theta'_{k+1}}{k+1} > \frac{\Theta_k}{k} \\
\Rightarrow & \text{lsbf}_{\phi'_{k+1}}(t) > \text{lsbf}_{\phi_k}(t) && \text{For all } t > 2(k-\Theta_k)
\end{aligned}
$$

This means that resource supply provided by model ϕ'_{k+1} is sufficient, but not necessary, to schedule component $\mathcal{C}$. Since ϕ_{k+1} is a minimum bandwidth resource model that can schedule $\mathcal{C}$, we then have

$$
\begin{aligned}
& \Theta'_{k+1} > \Theta_{k+1} \\
\Rightarrow & 2((k+1)-\Theta_{k+1}) > 2((k+1)-\Theta'_{k+1}) = 2(k-\Theta_k)
\end{aligned}
$$

We now show that resource bandwidths also increase with k, *i.e.*, $\frac{\Theta_{k+1}}{k+1} > \frac{\Theta_k}{k}$. To ensure schedulability of component $\mathcal{C}$ using resource model ϕ_{k+1} we require

$$
\begin{aligned}
& \text{lsbf}_{\phi_{k+1}}(t_k) \geq \text{dbf}_{\mathcal{C}}(t_k) = \text{lsbf}_{\phi_k}(t_k) \\
\Rightarrow & \frac{\Theta_{k+1}(t_k - 2(k+1-\Theta_{k+1}))}{k+1} \geq \frac{\Theta_k(t_k - 2(k-\Theta_k))}{k} && \text{Equation (2.4)} \\
\Rightarrow & \frac{\Theta_{k+1}}{k+1} > \frac{\Theta_k}{k} && \text{Since } \Theta'_{k+1} = \Theta_k + 1
\end{aligned}
$$

□

We now show that the intersection time instant of lsbf_{ϕ_k} with $\text{dbf}_{\mathcal{C}}$ and the $\text{dbf}_{\mathcal{C}}$ value at that time instant, are both non-increasing as period k increases.

Lemma A.2 *For each k, $1 \leq k < \Pi^*$, $t'_{k+1} \leq t_k$ and $\mathrm{dbf}_{\mathcal{C}}(t'_{k+1}) \leq \mathrm{dbf}_{\mathcal{C}}(t_k)$.*

Proof Let $\mathrm{dbf}_{\mathcal{C}}(t_k) = y_k$ and $\mathrm{dbf}_{\mathcal{C}}(t'_{k+1}) = y'_{k+1}$. We now assume that $y'_{k+1} > y_k$ and arrive at a contradiction. Since $\mathrm{lsbf}_{\phi_{k+1}}$ intersects $\mathrm{dbf}_{\mathcal{C}}$ at t'_{k+1}, we get

$$y'_{k+1} = \frac{\Theta_{k+1}(t'_{k+1} - 2(k+1-\Theta_{k+1}))}{k+1} \tag{A.1}$$

Also, since $\mathrm{lsbf}_{\phi_{k+1}}$ satisfies Equation (2.5), we have

$$y_k \leq \frac{\Theta_{k+1}(t_k - 2(k+1-\Theta_{k+1}))}{k+1} \tag{A.2}$$

Since $\mathrm{dbf}_{\mathcal{C}}$ is a non-decreasing function, $y'_{k+1} > y_k$ implies $t'_{k+1} > t_k$. Then, since $\mathrm{lsbf}_{\phi_k}(t_k) = y_k$ and $\mathrm{lsbf}_{\phi_k}(t'_{k+1}) \geq y'_{k+1}$,

$$\frac{y'_{k+1} - y_k}{t'_{k+1} - t_k} \leq \frac{\Theta_k}{k}$$

$\Rightarrow$(From Equations (A.1) and (A.2))

$$\frac{\Theta_{k+1}(t'_{k+1} - 2(k+1-\Theta_{k+1}))}{k+1} - \frac{\Theta_{k+1}(t_k - 2(k+1-\Theta_{k+1}))}{k+1} \leq \frac{(t'_{k+1} - t_k)\,\Theta_k}{k}$$

$$\Rightarrow \frac{\Theta_{k+1}(t'_{k+1} - t_k)}{k+1} \leq \frac{(t'_{k+1} - t_k)\,\Theta_{k+1}}{k+1}$$

$$\Rightarrow \frac{\Theta_{k+1}}{k+1} \leq \frac{\Theta_k}{k}$$

This contradicts Lemma A.1, and hence $y'_{k+1} \leq y_k$, which implies $\mathrm{dbf}_{\mathcal{C}}(t'_{k+1}) \leq \mathrm{dbf}_{\mathcal{C}}(t_k)$. Now, $\mathrm{dbf}_{\mathcal{C}}$ is a non-decreasing function of interval length. Also, for all $t > t_k$ such that t is a relevant time instant, $\mathrm{dbf}_{\mathcal{C}}(t) > \mathrm{dbf}_{\mathcal{C}}(t_k)$. Hence, we have also shown that $t'_{k+1} \leq t_k$. □

Since lsbf_{ϕ_1} intersects $\mathrm{dbf}_{\mathcal{C}}$ at t_1, using Lemma A.2 we get that for all k, $2 \leq k \leq \Pi^*$, $t_k \leq t'_k \leq t_1$. Thus, it is sufficient to only consider relevant time instants $\leq t_1$, which is what procedure SIG-INSTANTS does. We now show that the line segment connecting points $(t_k, \mathrm{dbf}_{\mathcal{C}}(t_k))$ and $(t'_{k+1}, \mathrm{dbf}_{\mathcal{C}}(t'_{k+1}))$ does not lie below any point $(t, \mathrm{dbf}_{\mathcal{C}}(t))$, where t is some relevant time instant smaller than t_k.

Lemma A.3 *Let $t(< t_k)$ denote some relevant time instant. Also, let $y = \mathrm{dbf}_{\mathcal{C}}(t), y_k = \mathrm{dbf}_{\mathcal{C}}(t_k)$ and $y'_{k+1} = \mathrm{dbf}_{\mathcal{C}}(t'_{k+1})$. Then, $t'_{k+1} \neq t_k$ implies $\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} \leq \frac{y_k - y}{t_k - t}$. Also, if $t'(< t'_{k+1})$ denotes some relevant time instant and $y' = \mathrm{dbf}_{\mathcal{C}}(t')$, then $t_{k+1} \neq t'_{k+1}$ implies $\frac{y'_{k+1} - y_{k+1}}{t'_{k+1} - t_{k+1}} \leq \frac{y'_{k+1} - y'}{t'_{k+1} - t'}$.*

Proof If $t = t'_{k+1}$ then the result is obvious. Hence, let $t \neq t'_{k+1}$. We assume that $\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} > \frac{y_k - y}{t_k - t}$ and arrive at a contradiction. Since lsbf_{ϕ_k} intersects $\mathrm{dbf}_{\mathcal{C}}$ at time instant t_k, we have

$$y_k = \frac{\Theta_k(t_k - 2(k - \Theta_k))}{k} \tag{A.3}$$

Similarly, since $\mathrm{lsbf}_{\phi_{k+1}}$ intersects $\mathrm{dbf}_{\mathcal{C}}$ at y'_{k+1}, we have

$$y'_{k+1} = \frac{\Theta_{k+1}(t'_{k+1} - 2(k + 1 - \Theta_{k+1}))}{k+1} \tag{A.4}$$

Furthermore, since lsbf_{ϕ_k} and $\mathrm{lsbf}_{\phi_{k+1}}$ both satisfy Equation (2.5), we get

$$y'_{k+1} \leq \frac{\Theta_k(t'_{k+1} - 2(k - \Theta_k))}{k} \tag{A.5}$$

$$y \leq \frac{\Theta_{k+1}(t - 2(k + 1 - \Theta_{k+1}))}{k+1} \tag{A.6}$$

Now, Equation (A.4) - Equation (A.6) gives

$$\frac{\Theta_{k+1}(t'_{k+1} - t)}{k+1} \leq y'_{k+1} - y \tag{A.7}$$

Also, Equation (A.5) - Equation (A.3) gives

$$y'_{k+1} - y_k \leq \frac{\Theta_k(t'_{k+1} - t_k)}{k} \tag{A.8}$$

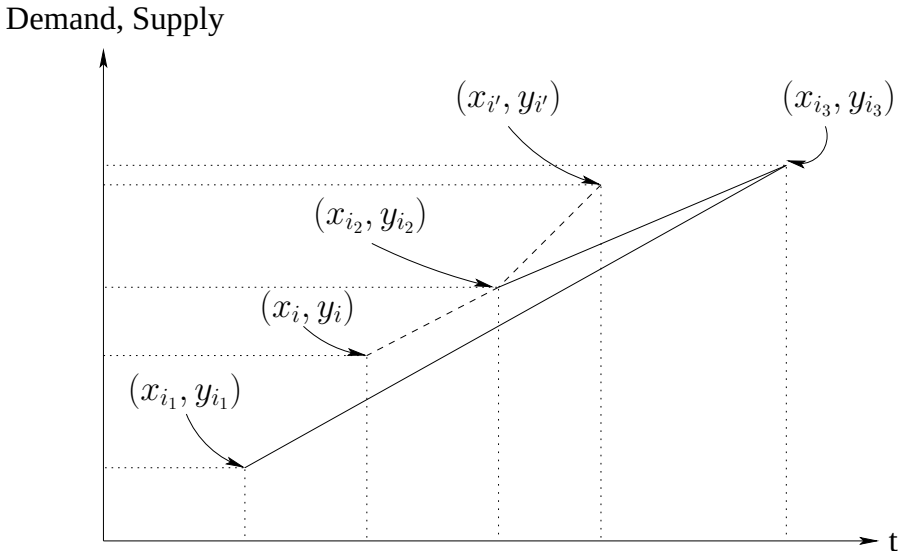


Figure A.1: Figure for Lemma A.4

Then,

$$\frac{y_k - y}{t_k - t} < \frac{y_k - y'_{k+1}}{t_k - t'_{k+1}}$$

$$\Rightarrow (y'_{k+1} - y)t_k < (y'_{k+1} - y_k)t + (y_k - y)t'_{k+1}$$

$$\Rightarrow (y'_{k+1} - y)t_k < (y'_{k+1} - y_k)t + ((y_k - y'_{k+1}) - (y - y'_{k+1}))t'_{k+1}$$

$$\Rightarrow (y'_{k+1} - y)(t_k - t'_{k+1}) < (y'_{k+1} - y_k)(t - t'_{k+1})$$

$\Rightarrow$(From Equations (A.7) and (A.8))

$$\frac{\Theta_{k+1}(t'_{k+1} - t)(t_k - t'_{k+1})}{k+1} < \frac{\Theta_k(t'_{k+1} - t_k)(t - t'_{k+1})}{k}$$

$$\Rightarrow \frac{\Theta_{k+1}(t - t'_{k+1})(t'_{k+1} - t_k)}{k+1} < \frac{\Theta_k(t'_{k+1} - t_k)(t - t'_{k+1})}{k}$$

$\Rightarrow$(Assuming $t'_{k+1} \neq t_k$)

$$\frac{\Theta_{k+1}}{k+1} < \frac{\Theta_k}{k}$$

This contradicts Lemma A.1, and hence we have shown that $t'_{k+1} \neq t_k$ implies $\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} \leq \frac{y_k - y}{t_k - t}$.

Second part of the lemma trivially follows from the facts $\text{lsbf}_{\phi_{k+1}}(t'_{k+1}) = y'_{k+1}$, $\text{lsbf}_{\phi_{k+1}}(t_{k+1}) = y_{k+1}$, $\text{lsbf}_{\phi_{k+1}}(t') \geq y'$, and $\text{lsbf}_{\phi_{k+1}}$ is linear. □

The following lemma proves a property of significant time instants generated by procedure SIG-INSTANTS. We use this property to show that the significant time instants subsume $t_1, \ldots, t_{\Pi^*}$ and $t'_2, \ldots, t'_{\Pi^*}$.

Lemma A.4 *Let x_{i_1}, x_{i_2} and x_{i_3} denote time instants in the list X passed to procedure* SIG-INSTANTS*, such that $x_{i_1} < x_{i_2} < x_{i_3}$. Further, let $y_{i_1} = \mathrm{dbf}_{\mathcal{C}}(x_{i_1}), y_{i_2} = \mathrm{dbf}_{\mathcal{C}}(x_{i_2})$ and $y_{i_3} = \mathrm{dbf}_{\mathcal{C}}(x_{i_3})$. If i_1 and i_3 denote adjacent elements in the index list $\mathcal{L}$ returned by procedure* SIG-INSTANTS*, then $\frac{y_{i_3}-y_{i_1}}{x_{i_3}-x_{i_1}} \leq \frac{y_{i_3}-y_{i_2}}{x_{i_3}-x_{i_2}}$.*

Proof We prove this lemma by contradiction. Assume i_1 and i_3 are adjacent indices in the list $\mathcal{L}$ and let

$$\begin{aligned} & \frac{y_{i_3} - y_{i_1}}{x_{i_3} - x_{i_1}} > \frac{y_{i_3} - y_{i_2}}{x_{i_3} - x_{i_2}} \\ \Rightarrow & \frac{y_{i_3} - y_{i_2}}{x_{i_3} - x_{i_2}} < \frac{y_{i_2} - y_{i_1}}{x_{i_2} - x_{i_1}} \qquad \text{Since } x_{i_3} > x_{i_2} > x_{i_1} \end{aligned} \tag{A.9}$$

Since i_1 and i_3 are adjacent in $\mathcal{L}$, i_2 was removed from $\mathcal{L}$ by procedure SIG-INSTANTS. Then, there exists indices i and i' such that, $x_{i'} > x_{i_2} > x_i$ and

$$\frac{y_{i'} - y_{i_2}}{x_{i'} - x_{i_2}} \geq \frac{y_{i_2} - y_i}{x_{i_2} - x_i}, \tag{A.10}$$

where $y_{i'} = \mathrm{dbf}_{\mathcal{C}}(x_{i'})$ and $y_i = \mathrm{dbf}_{\mathcal{C}}(x_i)$. Procedure SIG-INSTANTS checks this condition in Line 6. Now, we consider four cases and arrive at a contradiction in each of them.

Case 1: $i_3 = i', i_1 = i$

Contradiction is immediate from Equations (A.9) and (A.10).

Case 2: $i_3 = i', i_1 \neq i$

In this case we have

$$\frac{y_{i_3} - y_{i_2}}{x_{i_3} - x_{i_2}} \geq \frac{y_{i_2} - y_i}{x_{i_2} - x_i} \qquad \text{From Equation (A.10)}$$

$$\Rightarrow \frac{y_{i_3} - y_{i_2}}{x_{i_3} - x_{i_2}} \geq \frac{y_{i_3} - y_i}{x_{i_3} - x_i} \qquad \text{Since } x_{i_1} < x_i < x_{i_2} < x_{i_3}$$

$$\Rightarrow \frac{y_{i_3} - y_{i_1}}{x_{i_3} - x_{i_1}} > \frac{y_{i_3} - y_i}{x_{i_3} - x_i} \qquad \text{From assumption}$$

Now x_i has properties similar to x_{i_2}, and hence we can repeat these arguments for x_i. Since $x_i < x_{i_2}$ and number of relevant time instants in the interval (x_{i_1}, x_{i_3}) is finite, this case eventually reduces to Case 1.

Case 3: $i_3 \neq i', i_1 = i$

This case is similar to Case 2, and hence using similar arguments we can show that this case also reduces to Case 1.

Case 4: $i_3 \neq i', i_1 \neq i$

In this case, from Equations (A.9) and (A.10) we get that either $\frac{y_{i_3}-y_{i_1}}{x_{i_3}-x_{i_1}} > \frac{y_{i_3}-y_i}{x_{i_3}-x_i}$ or $\frac{y_{i_3}-y_{i_1}}{x_{i_3}-x_{i_1}} > \frac{y_{i_3}-y_{i'}}{x_{i_3}-x_{i'}}$ (see Figure A.1). Hence, Case 4 can be reduced to either Case 2 or Case 3.

Thus, we arrive at a contradiction in each of the four cases. □

We now show that the list of significant time instants generated by procedure SIG-INSTANTS and specified using list $\mathcal{L}$, contains all of $t_1, \ldots, t_{\Pi^*}$ and $t'_2, \ldots, t'_{\Pi^*}$.

Theorem A.5 *Let* $X = \langle x_1 = t_1, \ldots, x_N = \min_{i=1,\ldots,n}\{\mathrm{D}_i\}\rangle$ *denote the ordered list of relevant time instants, and* $Y = \langle y_1 = \mathrm{dbf}_{\mathcal{C}}(x_1), \ldots, y_N = \mathrm{dbf}_{\mathcal{C}}(x_N)\rangle$ *denote corresponding* dbf *values, passed to procedure* SIG-INSTANTS. *Furthermore, let* $\mathcal{L} = \langle l_1, \ldots, l_M\rangle$ *denote the index list returned by* SIG-INSTANTS. *Then,* $l_1 = 1, l_M = N$ *and for all* k, $1 \leq k < \Pi^*$, $t'_{k+1} \neq t_k$ *and* $t_k = x_{l_j}$ *implies* $t'_{k+1} = x_{l_{j+1}}$. *This implies* $x_l = t_{k+1}$ *for some* $l \in L$.

Proof List $\mathcal{L}$ is initialized to $\langle l_1 = 1, \ldots, l_N = N\rangle$ in Line 1 of procedure SIG-INSTANTS. In Line 7 the procedure removes an index from $\mathcal{L}$ that is equal to i_2. In this step i_1 is an index

smaller than i_2 and $i_1 \in \mathcal{L}$. Then $i_2 \neq 1$, and hence $l_1 = 1$ is never removed from $\mathcal{L}$. Using similar arguments we can show that $l_M = N$.

Now let $t'_{k+1} \neq t_k$ and $t_k = x_{l_j}$. We assume that $t'_{k+1} \neq x_{l_{j+1}}$ and arrive at a contradiction. We consider two cases depending on whether $t'_{k+1} > x_{l_{j+1}}$ or $t'_{k+1} < x_{l_{j+1}}$.

Case 1: $t'_{k+1} > x_{l_{j+1}}$

Since l_j and l_{j+1} are adjacent indices in $\mathcal{L}$ and $t_k > t'_{k+1} > x_{l_{j+1}}$, using Lemma A.4 we get

$$\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} \geq \frac{y_k - y_{l_{j+1}}}{t_k - x_{l_{j+1}}}$$

If this is a strict inequality ($>$), then it contradicts Lemma A.3. Otherwise,

$$\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} = \frac{y_k - y_{l_{j+1}}}{t_k - x_{l_{j+1}}}$$

and $\text{lsbf}_{\phi_{k+1}}$ intersects $\text{dbf}_{\mathcal{C}}$ at t_k, $x_{l_{j+1}}$, and t'_{k+1}. This contradicts our assumption that t'_{k+1} is the largest time instant at which $\text{lsbf}_{\phi_{k+1}}$ intersects $\text{dbf}_{\mathcal{C}}$.

Case 2: $t'_{k+1} < x_{l_{j+1}}$

Since $\min_{i=1,\ldots,n}\{\text{D}_i\} \leq t'_{k+1} < x_{l_{j+1}}$, there exists adjacent indices l_p and l_{p+1} in $\mathcal{L}$ such that $x_{l_{p+1}} \leq t'_{k+1} < x_{l_p}$. Then, from Lemma A.3 we get

$$\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} \leq \frac{y_k - y_{l_p}}{t_k - x_{l_p}}$$

and

$$\frac{y_k - y'_{k+1}}{t_k - t'_{k+1}} \leq \frac{y_k - y_{l_{p+1}}}{t_k - x_{l_{p+1}}}$$

Using the fact $x_{l_{p+1}} \leq t'_{k+1} < x_{l_p}$, we then get

$$\frac{y_{l_p} - y'_{k+1}}{x_{l_p} - t'_{k+1}} \leq \frac{y_{l_p} - y_{l_{p+1}}}{x_{l_p} - x_{l_{p+1}}}$$

If this inequality is strict ($<$), then it contradicts Lemma A.4. Otherwise,

$$\frac{y_{l_p} - y'_{k+1}}{x_{l_p} - t'_{k+1}} = \frac{y_{l_p} - y_{l_{p+1}}}{x_{l_p} - x_{l_{p+1}}},$$

and $\text{lsbf}_{\phi_{k+1}}$ intersects $\text{dbf}_{\mathcal{C}}$ at x_{l_p}, $x_{l_{p+1}}$, and t'_{k+1}. This contradicts our assumption that t'_{k+1} is the largest time instant at which $\text{lsbf}_{\phi_{k+1}}$ intersects $\text{dbf}_{\mathcal{C}}$.

Thus we have arrived at a contradiction in each of the two cases, and therefore $t'_{k+1} = x_{l_{j+1}}$. Then, using Lemmas A.3 and A.4 we also get that $x_l = t_{k+1}$ for some $l \in \mathcal{L}$. □

The following corollary then proves correctness of algorithm EDF-COMPACT.

Corollary A.6 *Let $\mathcal{I}_{\mathcal{C}}$ denote the multi-period interface that corresponds to the compact representation $\mathcal{CI}_{\mathcal{C}}$ generated by* EDF-COMPACT. *Then, $\mathcal{I}_{\mathcal{C}} = \bigcup_{k=1}^{\Pi^*} \phi_k$.*

Proof When period value is 1, EDF-COMPACT evaluates Equation (2.5) at all relevant time instants (Line 1 of the algorithm). Hence $\phi_1 \in \mathcal{I}_{\mathcal{C}}$.

For higher period values, EDF-COMPACT computes the periodic resource model by evaluating Equation (2.5) at two adjacent points in X, such that the points are specified by list $\mathcal{L}$. We now show that for each period value $k > 1$, EDF-COMPACT evaluates Equation (2.5) at time instant t'_k. Let $\mathcal{L} = \langle l_1, \ldots l_M \rangle$ denote the index list returned by procedure SIG-INSTANTS. For period value 2, EDF-COMPACT evaluates Equation (2.5) at two time instants; $csi = x_{l_1} = t_1$ and $nsi = x_{l_2}$ (Lines 14–25). In Line 16, $\Theta''_2 \leq \Theta'_2$ implies that the bandwidth of resource model with $\text{lsbf}(x_{l_2}) = \text{dbf}_{\mathcal{C}}(x_{l_2})$, is smaller than the bandwidth of resource model with $\text{lsbf}(t_1) = \text{dbf}_{\mathcal{C}}(t_1)$. In this case, from Theorem A.5 we get that lsbf_{ϕ_2} must intersect $\text{dbf}_{\mathcal{C}}$ at t_1 (*i.e.*, $t'_2 = t_1$). Hence, algorithm EDF-COMPACT proceeds without adding any tuple to $\mathcal{CI}_{\mathcal{C}}$. On the other hand, if $\Theta''_2 > \Theta'_2$, then using similar arguments and from Theorem A.5 we get that $t'_2 = x_{l_2}$. In this case the algorithm stores a new tuple in interface $\mathcal{CI}_{\mathcal{C}}$, and also updates counters csi and nsi to x_{l_2} and x_{l_3} respectively. The while loop in EDF-COMPACT then adjusts csi to coincide with t_2. Using similar arguments for all period values in the range 3 to Π^*, we get that EDF-COMPACT evaluates Equation (2.5) at time instant t'_k for each period k. Therefore, $\mathcal{I}_k = \bigcup_{k=1}^{\Pi^*} \phi_k$. □

Appendix B

Proof of correctness of algorithm DM-TASK

In this section we prove correctness of algorithm DM-TASK that we presented in Section 4.5. For this purpose we consider a component $\mathcal{C} = \langle \mathcal{T}, \text{DM} \rangle$, where $\mathcal{T} = \{\tau_1 = (\text{T}_1, \text{C}_1, \text{D}_1), \ldots, \tau_n = (\text{T}_n, \text{C}_n, \text{D}_n)\}$ denotes a set of constrained deadline sporadic tasks. We show that interface $\mathcal{CI}_{\mathcal{C},i}$ generated by the algorithm consists of minimum bandwidth resource models $\phi_{k,i} = \langle k, \Theta_{k,i} \rangle$, where k ranges from 1 to Π^*. In other words, we show that $\mathcal{I}_{\mathcal{C},i} = \bigcup_{k=1}^{\Pi^*} \phi_{k,i}$. For each k, we assume that $\text{lsbf}_{\phi_{k,i}}$ intersects $\text{rbf}_{\mathcal{C},i}$ at time instants t_k and t'_k, such that t_k denotes the largest and t'_k denotes the smallest such time instant.

Consider capacity $\Theta_{1,i}$ generated in Line 1 of DM-TASK. If $\Theta_{1,i} = 1$, then the minimum resource bandwidth required to schedule task τ_i is one. In this case $\text{lsbf}_{\phi_{k,i}}$ for all k are identical (slope one and blackout interval length zero), *i.e.*, $t_k = t_1$ for all k. DM-TASK then correctly computes interface $\mathcal{I}_{\mathcal{C},i}$ because t_1 is a significant time instant. In the remainder of this appendix we assume that $\frac{\Theta_{k,i}}{k} < 1$ for all k. We now state several lemmas in order to prove correctness of algorithm DM-TASK. These are similar to the lemmas that we proved for the EDF case.

Lemma B.1 states the monotonicity property of blackout interval and resource bandwidth.

Lemma B.1 *For each k, $1 \leq k < \Pi^*$, $2(k+1-\Theta_{k+1,i}) > 2(k-\Theta_{k,i})$ and $\frac{\Theta_{k+1,i}}{k+1} > \frac{\Theta_{k,i}}{k}$.*

Proof Similar to the proof of Lemma A.1 in Appendix A. □

Lemmas B.2, B.4, and B.3 prove correctness of procedure SIG-INSTANTS in the DM case.

Lemma B.2 *For each k, $1 \leq k < \Pi^*$, $t'_{k+1} \geq t_k$ and $\text{rbf}_{\mathcal{C},i}(t'_{k+1}) \geq \text{rbf}_{\mathcal{C},i}(t_k)$.*

Proof Similar to the proof of Lemma A.2 in Appendix A. □

Lemma B.3 *Let $t(> t_k)$ denote some relevant time instant. Also, let $y = \text{rbf}_{\mathcal{C},i}(t), y_k = \text{rbf}_{\mathcal{C},i}(t_k)$ and $y'_{k+1} = \text{rbf}_{\mathcal{C},i}(t'_{k+1})$. Then, $t'_{k+1} \neq t_k$ implies $\frac{y'_{k+1}-y_k}{t'_{k+1}-t_k} \leq \frac{y-y_k}{t-t_k}$. Also, if $t'(> t'_{k+1})$ denotes some relevant time instant and $y' = \text{rbf}_{\mathcal{C},i}(t')$, then $t_{k+1} \neq t'_{k+1}$ implies $\frac{y_{k+1}-y'_{k+1}}{t_{k+1}-t'_{k+1}} \leq \frac{y'-y'_{k+1}}{t'-t'_{k+1}}$.*

Proof Similar to the proof of Lemma A.3 in Appendix A. □

Lemma B.4 *Let x_{i_1}, x_{i_2}, and x_{i_3} denote time instants in the list X passed to procedure* SIG-INSTANTS *such that $x_{i_1} < x_{i_2} < x_{i_3}$. Further, let $y_{i_1} = \text{rbf}_{\mathcal{C},i}(x_{i_1}), y_{i_2} = \text{rbf}_{\mathcal{C},i}(x_{i_2})$, and $y_{i_3} = \text{rbf}_{\mathcal{C},i}(x_{i_3})$. If i_1 and i_3 denote adjacent elements in the index list $\mathcal{L}$ returned by* SIG-INSTANTS, *then $\frac{y_{i_3}-y_{i_1}}{x_{i_3}-x_{i_1}} \leq \frac{y_{i_2}-y_{i_1}}{x_{i_2}-x_{i_1}}$.*

Proof Similar to the proof of Lemma A.4 in Appendix A. □

Combining the results of these lemmas, we can show that Theorem A.5 given in Appendix A holds under DM scheduler as well. Then, the following corollary states correctness of algorithm DM-TASK.

Corollary B.5 *Let $\mathcal{I}_{\mathcal{C},i}$ denote the interface that corresponds to the compact representation $\mathcal{CI}_{\mathcal{C},i}$ generated by algorithm* DM-TASK *for task τ_i. Then, $\mathcal{I}_{\mathcal{C},i} = \bigcup_{k=1}^{\Pi^*} \phi_{k,i}$.*

Proof Similar to proof of Corollary A.6 in Appendix A. □

Appendix C

ARINC-653 workloads

C.1 Workloads with non-zero offset

Workload 1:

Partition name	Utilization
P1	0.134
P2	0.056
P3	0.028
P4	0.1265
P5	0.0335

Table C.1: Utilization for partitions in workload 1

```
<system os-scheduler="DM" >
  <component max-period="25" min-period="25" scheduler="DM" name=''P1'' >
    <task offset="2" jitter=''0" period="25" capacity="1.4" deadline="25" />
    <task offset="3" jitter=''0" period="50" capacity="3.9" deadline="50" />
  </component>
  <component max-period="50" min-period="50" scheduler="DM" name=''P2'' >
    <task offset="0" jitter=''0" period="50" capacity="2.8" deadline="50" />
  </component>
  <component max-period="50" min-period="50" scheduler="DM" name=''P3'' >
    <task offset="0" jitter=''0" period="50" capacity="1.4" deadline="50" />
  </component>
  <component max-period="25" min-period="25" scheduler="DM" name=''P4'' >
    <task offset="3" jitter=''0" period="25" capacity="1.1" deadline="25" />
    <task offset="5" jitter=''0" period="50" capacity="1.8" deadline="50" />
    <task offset="11" jitter=''0" period="100" capacity="2" deadline="100" />
    <task offset="13" jitter=''0" period="200" capacity="5.3" deadline="200" />
  </component>
  <component max-period="50" min-period="50" scheduler="DM" name=''P5'' >
    <task offset="2" jitter=''0" period="50" capacity="1.3" deadline="50" />
    <task offset="14" jitter=''0" period="200" capacity="1.5" deadline="200" />
  </component>
```

```
</system>
```

Workload 2:

Partition name	Utilization
P6	0.12
P7	0.1345
P8	0.165
P9	0.006
P10	0.038
P11	0.048

Table C.2: Utilization for partitions in workload 2

```
<system os-scheduler="DM" >
  <component max-period="50" min-period="50" scheduler="DM" name=''P6'' >
    <task offset="3" jitter=''0" period="50" capacity="5.4" deadline="50" />
    <task offset="15" jitter=''0" period="200" capacity="2.4" deadline="200" />
  </component>
  <component max-period="50" min-period="50" scheduler="DM" name=''P7'' >
    <task offset="1" jitter=''0" period="50" capacity="3.7" deadline="50" />
    <task offset="3" jitter=''0" period="100" capacity="1.8" deadline="100" />
    <task offset="4" jitter=''0" period="200" capacity="8.5" deadline="200" />
  </component>
  <component max-period="25" min-period="25" scheduler="DM" name=''P8'' >
    <task offset="2" jitter=''0" period="25" capacity="2.3" deadline="25" />
    <task offset="7" jitter=''0" period="100" capacity="4.8" deadline="100" />
    <task offset="9" jitter=''0" period="200" capacity="5" deadline="200" />
  </component>
  <component max-period="100" min-period="100" scheduler="DM" name=''P9'' >
    <task offset="0" jitter=''0" period="100" capacity="0.6" deadline="100" />
  </component>
  <component max-period="50" min-period="50" scheduler="DM" name=''P10'' >
    <task offset="0" jitter=''0" period="50" capacity="1.9" deadline="50" />
  </component>
  <component max-period="50" min-period="50" scheduler="DM" name=''P11'' >
    <task offset="0" jitter=''0" period="50" capacity="2.4" deadline="50" />
  </component>
</system>
```

C.2 Workloads with non-zero jitter

Workload 3:

```
<system os-scheduler="DM" >
  <component max-period="200000" min-period="200000" scheduler="DM" name="PART16 ID=16" vmips="0.8" >
    <task offset="0" jitter="1000" period="200000" capacity="282" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="863" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="701" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="106" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1370" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="607" deadline="200000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART29 ID=29" vmips="6.69" >
```

Partition name	Utilization	Reserved bandwidth
PART16 ID=16	0.019645	0.04505
PART29 ID=29	0.199415	0.37669
PART35 ID=35	0.05168	0.22185
PART20 ID=20	0.035125	0.09798
PART32 ID=32	0.033315	0.08164
PART36 ID=36	0.045	0.11036
PART33 ID=33	0.0379	0.09178
PART34 ID=34	0.04764	0.10755
PART17 ID=17	0.00408	0.01126
PART31 ID=31	0.00684	0.01689

Table C.3: Utilization and reserved bandwidth for partitions in workload 3

```
    <task offset="0" jitter="1000" period="25000" capacity="2260" deadline="25000" />
    <task offset="0" jitter="1000" period="200000" capacity="1643" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1158" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1108" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1108" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1108" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="6078" deadline="200000" />
    <task offset="0" jitter="1000" period="100000" capacity="4800" deadline="100000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART35 ID=35" vmips="3.94" >
    <task offset="0" jitter="1000" period="50000" capacity="1202" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="390" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="992" deadline="50000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART20 ID=20" vmips="1.74" >
    <task offset="0" jitter="1000" period="25000" capacity="290" deadline="25000" />
    <task offset="0" jitter="1000" period="100000" capacity="640" deadline="100000" />
    <task offset="0" jitter="1000" period="50000" capacity="675" deadline="50000" />
    <task offset="0" jitter="1000" period="200000" capacity="725" deadline="200000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART32 ID=32" vmips="1.45" >
    <task offset="0" jitter="1000" period="50000" capacity="1108" deadline="50000" />
    <task offset="0" jitter="5000" period="50000" capacity="218" deadline="50000" />
    <task offset="0" jitter="5000" period="200000" capacity="1359" deadline="200000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART36 ID=36" vmips="1.96" >
    <task offset="0" jitter="1000" period="200000" capacity="1000" deadline="200000" />
    <task offset="0" jitter="1000" period="25000" capacity="1000" deadline="25000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART33 ID=33" vmips="1.63" >
    <task offset="0" jitter="1000" period="50000" capacity="406" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="1352" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="137" deadline="50000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART34 ID=34" vmips="1.91" >
    <task offset="0" jitter="1000" period="50000" capacity="286" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="1870" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="226" deadline="50000" />
  </component>
  <component max-period="100000" min-period="100000" scheduler="DM" name="PART17 ID=17" vmips="0.2" >
    <task offset="0" jitter="1000" period="100000" capacity="408" deadline="100000" />
  </component>
  <component max-period="100000" min-period="100000" scheduler="DM" name="PART31 ID=31" vmips="0.3" >
    <task offset="0" jitter="1000" period="100000" capacity="684" deadline="100000" />
  </component>
</system>
```

Workload 4:

Partition name	Utilization	Reserved bandwidth
PART30 ID=30	0.11225	0.23086
PART16 ID=16	0.019645	0.04505
PART20 ID=20	0.035125	0.09797
PART17 ID=17	0.00408	0.01126
PART26 ID=26	0.13496	0.44932
PART27 ID=27	0.02784	0.06869
PART28 ID=28	0.055205	0.12106

Table C.4: Utilization and reserved bandwidth for partitions in workload 4

```
<system os-scheduler="DM" >
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART30 ID=30" vmips="4.1" >
    <task offset="0" jitter="1000" period="200000" capacity="2450" deadline="200000" />
    <task offset="0" jitter="1000" period="50000" capacity="5000" deadline="50000" />
  </component>
  <component max-period="200000" min-period="200000" scheduler="DM" name="PART16 ID=16" vmips="0.8" >
    <task offset="0" jitter="1000" period="200000" capacity="282" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="863" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="701" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="106" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1370" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="607" deadline="200000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART20 ID=20" vmips="1.74" >
    <task offset="0" jitter="1000" period="25000" capacity="290" deadline="25000" />
    <task offset="0" jitter="1000" period="100000" capacity="640" deadline="100000" />
    <task offset="0" jitter="1000" period="50000" capacity="675" deadline="50000" />
    <task offset="0" jitter="1000" period="200000" capacity="725" deadline="200000" />
  </component>
  <component max-period="100000" min-period="100000" scheduler="DM" name="PART17 ID=17" vmips="0.2" >
    <task offset="0" jitter="1000" period="100000" capacity="408" deadline="100000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART26 ID=26" vmips="7.98" >
    <task offset="0" jitter="1000" period="25000" capacity="1403" deadline="25000" />
    <task offset="0" jitter="1000" period="0" capacity="14783" deadline="0" />
    <task offset="0" jitter="1000" period="50000" capacity="3942" deadline="50000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART27 ID=27" vmips="1.22" >
    <task offset="0" jitter="1000" period="50000" capacity="1391" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="1" deadline="50000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART28 ID=28" vmips="2.15" >
    <task offset="0" jitter="1000" period="50000" capacity="2760" deadline="50000" />
    <task offset="0" jitter="1000" period="200000" capacity="1" deadline="200000" />
  </component>
</system>
```

Workload 5:

```
<system os-scheduler="DM" >
  <component max-period="6250" min-period="6250" scheduler="DM" name="PART15 ID=15" vmips="0" >
    <task offset="0" jitter="10" period="6250" capacity="3255" deadline="6250" />
    <task offset="0" jitter="1000" period="200000" capacity="0" deadline="200000" />
    <task offset="0" jitter="1000" period="100000" capacity="0" deadline="100000" />
    <task offset="0" jitter="1000" period="25000" capacity="0" deadline="25000" />
    <task offset="0" jitter="1000" period="50000" capacity="0" deadline="50000" />
```

Partition name	Utilization	Reserved bandwidth
PART15 ID=15	0.5208	0
PART13 ID=13	0.01126	0.03378
PART12 ID=12	0.0050	0.01126

Table C.5: Utilization and reserved bandwidth for partitions in workload 5

```
  </component>
  <component max-period="200000" min-period="200000" scheduler="DM" name="PART13 ID=13" vmips="0.6" >
    <task offset="0" jitter="1000" period="200000" capacity="282" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="863" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="500" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="607" deadline="200000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART12 ID=12" vmips="0.2" >
    <task offset="0" jitter="1000" period="100000" capacity="500" deadline="100000" />
    <task offset="0" jitter="1000" period="25000" capacity="0" deadline="25000" />
  </component>
</system>
```

Workload 6:

Partition name	Utilization	Reserved bandwidth
PART16 ID=16	0.019645	0.04505
PART19 ID=19	0.14008	0.32939
PART21 ID=21	0.12751	0.30011
PART22 ID=22	0.13477	0.31137
PART17 ID=17	0.00408	0.01126

Table C.6: Utilization and reserved bandwidth for partitions in workload 6

```
<system os-scheduler="DM" >
  <component max-period="200000" min-period="200000" scheduler="DM" name="PART16 ID=16" vmips="0.8" >
    <task offset="0" jitter="1000" period="200000" capacity="282" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="863" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="701" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="106" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="1370" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="607" deadline="200000" />
  </component>
  <component max-period="12500" min-period="12500" scheduler="DM" name="PART19 ID=19" vmips="5.85" >
    <task offset="0" jitter="1000" period="12500" capacity="645" deadline="12500" />
    <task offset="0" jitter="1000" period="100000" capacity="1565" deadline="100000" />
    <task offset="0" jitter="1000" period="50000" capacity="1440" deadline="50000" />
    <task offset="0" jitter="1000" period="25000" capacity="1010" deadline="25000" />
    <task offset="0" jitter="1000" period="200000" capacity="725" deadline="200000" />
  </component>
  <component max-period="25000" min-period="25000" scheduler="DM" name="PART21 ID=21" vmips="5.22" >
    <task offset="0" jitter="1000" period="25000" capacity="217" deadline="25000" />
    <task offset="0" jitter="100" period="25000" capacity="840" deadline="25000" />
    <task offset="0" jitter="1000" period="50000" capacity="1944" deadline="50000" />
    <task offset="0" jitter="1000" period="100000" capacity="1988" deadline="100000" />
    <task offset="0" jitter="1000" period="200000" capacity="5294" deadline="200000" />
  </component>
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART22 ID=22" vmips="5.53" >
```

```
    <task offset="0" jitter="1000" period="50000" capacity="238" deadline="50000" />
    <task offset="0" jitter="1000" period="50000" capacity="3450" deadline="50000" />
    <task offset="0" jitter="1000" period="100000" capacity="1868" deadline="100000" />
    <task offset="0" jitter="1000" period="200000" capacity="8466" deadline="200000" />
    <task offset="0" jitter="1000" period="0" capacity="1855" deadline="0" />
  </component>
  <component max-period="100000" min-period="100000" scheduler="DM" name="PART17 ID=17" vmips="0.2" >
    <task offset="0" jitter="1000" period="100000" capacity="408" deadline="100000" />
  </component>
</system>
```

Workload 7:

Partition name	Utilization	Reserved bandwidth
PART45 ID=45	0.00325	0.02815

Table C.7: Utilization and reserved bandwidth for partitions in workload 7

```
<system os-scheduler="DM" >
  <component max-period="50000" min-period="50000" scheduler="DM" name="PART45 ID=45" vmips="0.5" >
    <task offset="0" jitter="1000" period="200000" capacity="400" deadline="200000" />
    <task offset="0" jitter="1000" period="200000" capacity="50" deadline="200000" />
    <task offset="0" jitter="1000" period="50000" capacity="50" deadline="50000" />
  </component>
</system>
```

Bibliography

[1] Tarek Abdelzaher and Kang Shin. Optimal Combined Task and Message Scheduling in Distributed Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 162–171, 1995.

[2] Luis Almeida and Paulo Pedreiras. Scheduling within Temporal Partitions: Response-Time Analysis and Server Design. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 95–103, 2004.

[3] Madhukar Anand, Arvind Easwaran, Sebastian Fischmeister, and Insup Lee. Compositional Feasibility Analysis for Conditional Real-Time Task Models. In *IEEE International Symposium on Object/Component/Service-oriented Real-Time Distributed Computing*, pages 391–398, 2008.

[4] James Anderson, John Calandrino, and UmaMaheswari Devi. Real-Time Scheduling on Multicore Platforms. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 179–190, 2006.

[5] James Anderson and Anand Srinivasan. Early-Release Fair Scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 35–43, 2000.

[6] Björn Andersson, Sanjoy Baruah, and Jan Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 193–202, 2001.

[7] Björn Andersson and Konstantinos Bletsas. Sporadic Multiprocessor Scheduling with Few Preemptions. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 243–252, 2008.

[8] Björn Andersson, Konstantinos Bletsas, and Sanjoy Baruah. Scheduling Arbitrary-Deadline Sporadic Tasks on Multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 387–396, 2008.

[9] Björn Andersson and Eduardo Tovar. Multiprocessor Scheduling with Few Preemptions. In *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

[10] Neil Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Technical Report YCS164, Department of Computer Science, University of York, 1991.

[11] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 225–232, 2001.

[12] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 95–105, 2001.

[13] Theodore Baker. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[14] Theodore Baker. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):760–768, 2005.

[15] Theodore Baker. Comparison of Empirical Success Rates of Global vs. Partitioned Fixed-Priority EDF Scheduling for Hard Real-Time. Technical Report TR–050601, Department of Computer Science, Florida State University, Tallahassee, 2005.

[16] Theodore Baker. An Analysis of Fixed-Priority Schedulability on a Multiprocessor. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 32(1–2):49–71, 2006.

[17] Sanjoy Baruah. Feasibility Analysis of Recurring Branching Tasks. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 138–145, 1998.

[18] Sanjoy Baruah. A general model for recurring real-time tasks. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 114–122, 1998.

[19] Sanjoy Baruah. Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, 2004.

[20] Sanjoy Baruah. Techniques for Multiprocessor Global Schedulability Analysis. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 119–128, 2007.

[21] Sanjoy Baruah and Theodore Baker. Global EDF Schedulability Analysis of Arbitrary Sporadic Task Systems. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 3–12, 2008.

[22] Sanjoy Baruah and Theodore Baker. Schedulability Analysis of Global EDF. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 38(3):223–235, 2008.

[23] Sanjoy Baruah and John Carpenter. Multiprocessor Fixed-Priority Scheduling with Restricted Interprocessor Migrations. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 169–178, 2003.

[24] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized Multiframe Tasks. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 17(1):5–22, 1999.

[25] Sanjoy Baruah, N. K. Cohen, C. Greg Plaxton, and Donald Varvel. Proportionate Progress: A Notion of Fairness in Resource Allocation. 15(6):600–625, 1996.

[26] Sanjoy Baruah and Nathan Fisher. The Partitioned Multiprocessor Scheduling of Deadline-Constrained Sporadic Task Systems. *IEEE Transactions on Computers*, 55(7):918–923, 2006.

[27] Sanjoy Baruah and Nathan Fisher. Global Deadline-Monotonic Scheduling of Arbitrary-Deadline Sporadic Task Systems. In *International Conference on Principles of Distributed Systems*, pages 204–216, 2007.

[28] Sanjoy Baruah, Rodney Howell, and Louis Rosier. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 2(4):301–324, 1990.

[29] Sanjoy Baruah, Aloysius Mok, and Louis Rosier. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

[30] Moris Behnam, Insik Shin, Thomas Nolte, and Mikael Nolin. SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 279–288, 2007.

[31] Marko Bertogna and Michele Cirinei. Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 149–160, 2007.

[32] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Improved Schedulability Analysis of EDF on Multiprocessor Platforms. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 209–218, 2005.

[33] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New Schedulability Tests for Real-Time Task Sets Scheduled by Deadline Monotonic on Multiprocessors. In *International Conference on Principles of Distributed Systems*, pages 306–321, 2005.

[34] Riccardo Bettati and J. W. S. Liu. End-to-End Scheduling to Meet Deadlines in Distributed Systems. In *Proceedings of IEEE International Conference on Distributed Computing Systems*, pages 452–459, 1992.

[35] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX (Third Edition)*. Addison Wesley Longmain, 2001.

[36] José Busquets-Mataix, Juan Serrano, Rafael Ors, Pedro Gil, and Andy Wellings. Using harmonic task-sets to increase the schedulable utilization of cache-based preemptive real-time systems.

[37] John Calandrino, James Anderson, and Dan Baumberger. A Hybrid Real-Time Scheduling Approach for Large-scale Multicore Platforms. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 247–258, 2007.

[38] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 101–110, 2006.

[39] S. Cho, S. K. Lee, S. Ahn, and K. J. Lin. Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. *IEICE Transactions on Communications*, E85–B(12):2859–2867, 2002.

[40] Michele Cirinei and Theodore Baker. EDZL Scheduling Analysis. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 9–18, 2007.

[41] Citrix Systems Inc. Virtualization Platform. www.citrixxenserver.com.

[42] Robert Davis and Alan Burns. Hierarchical Fixed Priority Pre-emptive Scheduling. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 389–398, 2005.

[43] Robert Davis and Alan Burns. Resource Sharing in Hierarchical Fixed Priority Pre-emptive Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 257–270, 2006.

[44] Luca de Alfaro and Thomas Henzinger. Interface Automata. In *Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 109–120, 2001.

[45] Luca de Alfaro and Thomas Henzinger. Interface Theories for Component-based Design. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 148–165, 2001.

[46] Luca de Alfaro, Thomas Henzinger, and Marielle Stoelinga. Timed Interfaces. In *Proceedings of Second International Workshop on Embedded Software*, pages 101–122, 2002.

[47] Zhong Deng and J. W. S. Liu. Scheduling Real-Time Applications in an Open Environment. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 308–319, 1997.

[48] Michael Dertouzos. Control Robotics: The Procedural Control of Physical Processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[49] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional Analysis Framework using EDP Resource Models. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 129–138, 2007.

[50] Arvind Easwaran, Madhukar Anand, Insup Lee, and Oleg Sokolsky. On the Complexity of Generating Optimal Interfaces for Hierarchical Systems. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS)*, 2008.

[51] Arvind Easwaran, Insup Lee, Insik Shin, and Oleg Sokolsky. Compositional Schedulability Analysis of Hierarchical Real-Time Systems. In *IEEE International Symposium on Object/Component/Service-oriented Real-Time Distributed Computing*, pages 274–281, 2007.

[52] Arvind Easwaran, Insup Lee, Oleg Sokolsky, and Steve Vestal. A Compositional Framework for Avionics (ARINC-653) Systems. In *Under submission*, 2008.

[53] Arvind Easwaran, Insik Shin, Insup Lee, and Oleg Sokolsky. Bounding Preemptions under EDF and RM Schedulers. Technical Report MS–CIS–06–06, Department of Computer and Information Science, University of Pennsylvania, 2006.

[54] Arvind Easwaran, Insik Shin, Oleg Sokolsky, and Insup Lee. Incremental Schedulability Analysis of Hierarchical Real-Time Components. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 272–281, 2006.

[55] Engineering Standards for Avionics and Cabin Systems (AEEC). *ARINC Specification 653-2, Part I*, 2006.

[56] Xiang Feng and Aloysius Mok. A Model of Hierarchical Real-Time Virtual Resources. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 26–35, 2002.

[57] Nathan Fisher, Sanjoy Baruah, and Theodore Baker. The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 118–127, 2006.

[58] Nathan Fisher, Marko Bertogna, and Sanjoy Baruah. The Design of an EDF-Scheduled Resource-Sharing Open Environment. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 83–92, 2007.

[59] Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki. Work-Conserving Optimal Real-Time Scheduling on Multiprocessors. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 13–22, 2008.

[60] Jose Garcia and Michael Harbour. Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems. In *Proceedings of IEEE Workshop on Parallel and Distributed Real-Time Systems*.

[61] Richard Gerber, Seongsoo Hong, Manas Saksena, and Dong in Kang. End-to-End Design of Real-Time Systems. Technical Report UMIACS-TR-95-61, University of Maryland, Institute for Advanced Computer Studies, 1995.

[62] Joel Goossens, Shelby Funk, and Sanjoy Baruah. Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 25(2–3):187–205, 2003.

[63] Gregor Gössler and Joseph Sifakis. Composition for Component-based Modeling. *Science of Computer Programming*, 55(1–3):161–183, 2005.

[64] Ronald Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972.

[65] Green Hills Software Inc. ARINC 653 Partition Scheduler. www.ghs.com/products/safety_critical/arinc653.html.

[66] Thomas Henzinger, Benjamin Horowitz, and Christoph Kirsch. GIOTTO: A Time-Triggered Language for Embedded Programming. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 166–184, 2001.

[67] Thomas Henzinger and Slobodan Matic. An Interface Algebra for Real-Time Components. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 253–266, 2006.

[68] Philip Holman and James Anderson. Guaranteeing Pfair Supertasks by Reweighting. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 203–212, 2001.

[69] Jan Jonsson and Kang Shin. Deadline Assignment in Distributed Hard Real-Time Systems with Relaxed Locality Constraints. In *Proceedings of IEEE International Conference on Distributed Computing Systems*, pages 432–440, 1997.

[70] George Thomas Jr. and Ross Finney. *Calculus and Analytic Geometry (9th edition)*. Addison Wesley, 1996.

[71] Shinpei Kato and Nobuyuki Yamasaki. Real-Time Scheduling with Task Splitting on Multiprocessors. In *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 441–450, 2007.

[72] Larry Kinnan, Joseph Wlad, and Patrick Rogers. Porting Applications to an ARINC 653 Compliant IMA Platform Using VXworks as an Example. In *Proceedings of IEEE Digital Avionics Systems Conference*, pages 10.B.1–10.1–8: Volume 2, 2004.

[73] Tei-Wei Kuo and Ching-Hui Li. A Fixed-Priority-driven Open Environment for Real-Time Applications. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 256–267, 1999.

[74] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou. Scheduling Tool and Algorithm for Integrated Modular Avionics Systems. In *Proceedings of IEEE Digital Avionics Systems Conference*, pages 1C2/1–1C2/8: Volume 1, 2000.

[75] John Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 201–209, 1990.

[76] John Lehoczky, Lui Sha, and Ye Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[77] Hennadiy Leontyev and James Anderson. A Hierarchical Multiprocessor Bandwidth Reservation Scheme with Timing Guarantees. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 191–200, 2008.

[78] J.Y.T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. *Performance Evaluation*, 2:237–250, 1982.

[79] Giuseppe Lipari and Sanjoy Baruah. Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 166–175, 2000.

[80] Giuseppe Lipari and Enrico Bini. Resource Partitioning Among Real-Time Applications. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 151–158, 2003.

[81] Giuseppe Lipari, John Carpenter, and Sanjoy Baruah. A Framework for Achieving Inter-Application Isolation in Multiprogrammed Hard-Real-Time Environments. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 217–226, 2000.

[82] C.L. Liu and James Layland. Scheduling Algorithms for Multi-Programming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[83] J. M. López, J. L. Díaz, and D. F. García. Minimum and Maximum Utilization Bounds for Multiprocessor RM Scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 67–75, 2001.

[84] Mathworks. Models with Multiple Sample Rates. Real-Time Workshop User Guide, 2005.

[85] Slobodan Matic and Thomas Henzinger. Trading End-to-End Latency for Composability. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 99–110, 2005.

[86] Robert McNaughton. Scheduling with Deadlines and Loss Functions. *Management Science*, 6(1):1–12, 1959.

[87] Mark Moir and Srikanth Ramamurthy. Pfair Scheduling of Fixed and Migrating Periodic Tasks on Multiple Resources. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 294–303, 1999.

[88] Aloysius Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

[89] Aloysius Mok and Deji Chen. A Multiframe Model for Real-Time Tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[90] Aloysius Mok, Xiang Feng, and Deji Chen. Resource Partition for Real-Time Systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 75–84, 2001.

[91] Aloysius Mok, Duu-Chung Tsou, and Ruud de Rooij. The MSP.RTL Real-Time Scheduler Synthesis Tool. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 118–128, 1996.

[92] Marco Di Natale and John Stankovic. Dynamic End-to-End Guarantees in Distributed Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 216–227, 1994.

[93] Dong-Ik Oh and Theodore Baker. Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 15(2):183–192, 1998.

[94] Harini Ramaprasad and Frank Mueller. Tightening the Bounds on Feasible Preemption Points. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 212–224, 2006.

[95] John Regehr and John Stankovic. HLS: A Framework for Composing Soft Real-Time Schedulers. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 3–14, 2001.

[96] Saowanee Saewong and Ragunathan Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 106–114, 2003.

[97] Saowanee Saewong, Ragunathan Rajkumar, John Lehoczky, and Mark Klein. Analysis of Hierarchical Fixed-Priority Scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 173–180, 2002.

[98] Manas Saksena and Seongsoo Hong. An Engineering Approach to Decomposing End-to-End Delays on a Distributed Real-Time System. In *Proceedings of IEEE Workshop on Parallel and Distributed Real-Time Systems*, pages 244–251, 1996.

[99] Wei Kuan Shih, J. W. S. Liu, and C. L. Liu. Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines. *IEEE Transactions on Software Engineering*, 19(12):1171–1179, 1993.

[100] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 181–190, 2008.

[101] Insik Shin and Insup Lee. Compositional Real-Time Scheduling Framework. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 57–67, 2004.

[102] Insik Shin and Insup Lee. Compositional Real-Time Scheduling Framework with Periodic Model. *ACM Transactions on Embedded Computing Systems*, 7(3), 2008.

[103] Marco Spuri. Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems. Technical Report RR-2873, INRIA: The French National Institute for Research in Computer Science and Control, 1996.

[104] Anand Srinivasan and James Anderson. Optimal Rate-based Scheduling on Multiprocessors. *Journal of Computer and System Sciences*, 72(6):1094–1117, 2006.

[105] Anand Srinivasan and Sanjoy Baruah. Deadline-based Scheduling of Periodic Task Systems on Multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

[106] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-Time Calculus for Scheduling Hard Real-Time Systems. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 101–104, 2000.

[107] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. Real-Time Interfaces for Composing Real-Time Systems. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 34–43, 2006.

[108] Ken Tindell. Adding Time-Offsets to Schedulability Analysis. Technical Report YCS 221, Department of Computer Science, University of York, 1994.

[109] Ken Tindell and John Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[110] Stephen Vavasis. *Complexity Issues in Global Optimization: A Survey.* Handbook of Global Optimization: Nonconvex Optimization and its Applications, Kluwer, 1995.

[111] Steve Vestal. Real-Time Sampled Signal Flows through Asynchronous Distributed Systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 170–179, 2005.

[112] VMware Inc. ESX Server. www.vmware.com/products/vi/esx/.

[113] Ernesto Wandeler and Lothar Thiele. Real-Time Interface for Interface-Based Design of Real-Time Systems with Fixed Priority Scheduling. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 80–89, 2005.

[114] Ernesto Wandeler and Lothar Thiele. Interface-Based Design of Real-Time Systems with Hierarchical Scheduling. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 80–89, 2006.

[115] Windriver Inc. Platform for ARINC 653. www.windriver.com/products/platforms/safety_critical/.

[116] Fengxiang Zhang and Alan Burns. Analysis of Hierarchical EDF Pre-emptive Scheduling. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 423–434, 2007.

[117] Dakai Zhu, Daniel Mossé, and Rami Melhem. Multiple-Resource Periodic Scheduling Problem: How Much Fairness is Necessary? In *Proceedings of IEEE Real-Time Systems Symposium*, pages 142–153, 2003.