EFFICIENT ALLOCATION AND ENFORCEMENT OF INTERFACES IN COMPOSITIONAL REAL-TIME SYSTEMS

by

FARHANA DEWAN

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2014

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor Date

_____

_____

_____

DEDICATION

to my

MOTHER and FATHER

with love

ACKNOWLEDGEMENTS

Firstly, I would like to express my deep-felt gratitude to my advisor, Dr. Nathan W. Fisher of the Computer Science Department at Wayne State University, for his constant support, advice, encouragement and enduring patience. He always believed in me even though I was often doubting in my own abilities. He provided clear explanations when I was lost, constantly drove me with energy when I was tired, and always give me his time and guidance, in spite of anything else that was going on. I am thankful to him for understanding me and being patient during my tough times. It was a great pleasure working with him as he is by far my best mentor.

I also wish to thank the members of my committee, Dr. Weisong Shi and Dr. Monica Brockmeyer of the Computer Science Department, Wayne State University, and Dr. Thomas Nolte of Mälardalen University, Sweden. Each of them inspired me to become successful in pursuing my research goals. Their suggestions, comments and additional guidance were invaluable to the completion of this work.

I want to thank my lab-mates for inspiring me and helping me whenever it was required. I want to thank Wayne State University Computer Science Department professors and staff for all their hard work and dedication, providing me the means to complete my degree. Additionally I am really grateful to all my friends at Wayne State and Detroit, who constantly supported and encouraged me with their time and advice. They were my second family while living in Detroit. I want to express sincere gratitude to my late friend Anupam Bhattacharjee, who were like a brother to me and helped me in every possible way.

I cannot thank enough to my mother Anowara Begum and father Lutfor Rahman Dewan for constantly loving, encouraging and supporting me in fulfilling my dreams. I am grateful to my brothers Syedur Rahaman Dewan and Rajib Dewan for always supporting my decisions and keeping my spirit up. I am thankful to all my family members in our home at Dhaka for their kindness and encouragement.

Finally, I would like to thank the funding authorities. The research contained in this dissertation has been supported in part by a National Science Foundation CAREER Award (CNS-0953585) and a grant from Wayne State University's Office of the Vice President for Research.

TABLE OF CONTENTS

Dedication ii

Acknowledgements iii

List of Tables ix

List of Figures x

List of Algorithms xii

List of Notations xiii

Chapter 1: Introduction 1

1.1 Compositional Real-Time System 2

1.1.1 Component-Based Design 3

1.1.2 Hierarchical Scheduling 4

1.1.3 Real-Time Interfaces 4

Server-Based Interface 5

Demand-Based Interface 5

1.1.4 Compositional Schedulability Analysis 6

1.2 Fundamental Problems in Compositional Systems 7

1.2.1 Allocation: Minimization of Interface Bandwidth 7

1.2.2 Enforcement: Admission Control 10

1.2.3 Slack Reclamation 11

1.3 Applications 12

1.3.1 Automotive 12

1.3.2 Avionics 13

1.3.3 Energy-Aware Systems 13

1.4 Contribution 14

1.5 Organization 16

Chapter 2: Related Work 17

2.1 Uniprocessor Platform 17

2.1.1 Server-Based Models . . . 17
Real-Time Open Environment . . . 18
Resource Kernel . . . 18
Bounded-Delay Model . . . 18
Periodic Resource Model . . . 19
Explicit Deadline Periodic Resource . . . 19
Fixed-Priority Servers . . . 20
2.1.2 Interface-Based Models . . . 21
Real-Time Calculus . . . 21
Bandwidth Sharing Server . . . 22
Demand-Bound Server . . . 22
2.1.3 Slack Reclamation . . . 23
2.2 Multiprocessor Platform . . . 24
Chapter 3: Model and Notations . . . 25
3.1 Task Model . . . 25
3.1.1 Aperiodic Job Model . . . 25
3.1.2 Sporadic Task Model . . . 26
3.2 Workload Functions . . . 26
3.2.1 Aperiodic Jobs . . . 27
3.2.2 Sporadic Tasks . . . 27
Earliest-Deadline First Scheduling . . . 27
Fixed-Priority Scheduling . . . 29
3.3 Resource Model . . . 30
3.3.1 Server-Based Interface . . . 30
3.3.2 Demand-Curve Interface . . . 32
Single Step Demand Interface . . . 32
Arbitrary Demand Interface . . . 33
Chapter 4: Allocation: Server-Based Model (EDF) . . . 34

4.1 MIB-RT in Periodic Resources for EDF-Scheduled Components 34
4.1.1 An Exact Solution 34
4.1.2 An Over-provisioned Solution 35
4.2 Approximate Solution 35
4.2.1 Approximate Capacity Determination 38
4.2.2 Algorithm Complexity 39
4.2.3 Algorithm Correctness 39
4.2.4 Approximation Ratio 49
4.3 Simulation Results 53
Chapter 5: Allocation: Server-Based Model (FP) 58
5.1 MIB-RT in Periodic Resources for FP-Scheduled Components 59
5.2 Determining Minimum Capacity Using Response Time 59
5.2.1 Deriving Response Time Lower Bound 61
5.2.2 Deriving Response Time Upper Bound 62
5.3 Determining Minimum Capacity Using Testing Set 63
5.3.1 Exact Capacity Determination 63
5.3.2 Approximate Capacity Determination 64
Algorithm Description 67
Algorithm Complexity 69
Algorithm Correctness 69
Approximation Ratio 85
5.3.3 Simulation Results 89
5.4 Efficient Capacity Determination for Arbitrary Deadline FP-Scheduled Components 93
5.4.1 Schedulability Algorithm for FP-Scheduled Arbitrary-Deadline Tasks 96
Algorithm Description 98
Algorithm Complexity 99
Algorithm Correctness 100
5.4.2 Simulation Results 107

Chapter 6: Enforcement: Interface-Based Model . . . 109
6.1 Enforcing Demand-Curve Interface for Aperiodic Workload . . . 109
6.2 Simple Demand-Curve Interface . . . 112
6.2.1 Exact Admission Control for MAD Jobs . . . 112
Algorithm Complexity . . . 113
Algorithm Correctness . . . 114
6.2.2 Exact Admission Control for Arbitrary Jobs . . . 114
Algorithm Complexity . . . 117
Algorithm Correctness . . . 117
6.3 Arbitrary Demand-Curve Interface . . . 117
6.3.1 Exact Admission Control for MAD Jobs . . . 118
Algorithm Complexity . . . 119
Algorithm Correctness . . . 120
6.3.2 Inadequacy of Naive Reduction in Intervals . . . 121
6.3.3 Approximate Admission Control for MAD Jobs . . . 123
Algorithm Correctness . . . 129
Approximation Ratio . . . 130
Algorithm Complexity . . . 132
6.4 Deriving Upper-bound for dbi . . . 134
6.4.1 Eliminating Approximation Points for Periodic dbi . . . 136
6.4.2 Server Design . . . 138
Temporal Isolation . . . 139
Resource Reclamation . . . 139
6.5 Resetting the Admission Controller . . . 140
6.5.1 Monotonic Increasing-Rate dbi . . . 141
6.6 Simulation Results . . . 143
6.7 Admission Controller for Arbitrary Aperiodic Jobs . . . 145
6.7.1 Algorithm Description . . . 146

6.7.2 Algorithm Correctness . . . 149
Chapter 7: Enforcement: Applying in Slack Reclamation . . . 153
7.1 Optimal Runtime Slack . . . 154
7.1.1 Task and Workload Models . . . 154
7.1.2 Runtime System Slack . . . 157
7.2 Determining Exact Runtime Slack . . . 159
7.2.1 Computing Slack Table . . . 159
7.2.2 Exact Algorithm . . . 160
7.2.3 Example . . . 160
7.3 Efficient Implementation of Slack Table . . . 161
7.3.1 Space-Filling Curves . . . 161
7.4 Determining Approximate Runtime Slack . . . 162
7.5 Simulation . . . 163
7.6 Discussion . . . 165
Chapter 8: Summary and Future Work . . . 166
8.1 Our Contribution . . . 166
8.2 Future Work . . . 167
8.2.1 Enforcing Demand-Curve Interface for Multiprocessor . . . 167
8.2.2 Implementation of Admission Controller in Operating System . . . 168
8.2.3 Further Extensions . . . 168
8.3 Concluding Remarks . . . 169
Appendix
Appendix A: Proofs from Chapter 6 . . . 170
References . . . 178
Abstract . . . 188
Autobiographical Statement . . . 191

LIST OF TABLES

Table 6.1 Periodic dbi parameters 143

Table 7.1 Computation of offset table 161

LIST OF FIGURES

Figure 1.1 Compositional real-time system 3

Figure 1.2 Real-time interfaces 6

Figure 1.3 An example of resource allocation in the EDP model. 8

Figure 1.4 MIB-RT problem for periodic resources 9

Figure 3.1 MAD jobs: j_i has greater absolute deadline than j_n 26

Figure 3.2 Arbitrary jobs: j_i has smaller absolute deadline than j_{n-1} 26

Figure 3.3 Demand-bound function $\mathsf{dbf}(\tau_i, t)$ of task τ_i 28

Figure 3.4 Request-bound function $\mathsf{rbf}(\tau_i, t)$ of task τ_i 28

Figure 3.5 Demand-curve interfaces 32

Figure 4.1 Supply-bound function with ℓ-feasibility region 40

Figure 4.2 Illustration of Lemma 3 42

Figure 4.3 Relative Error vs System Utilization 55

Figure 4.4 Testing Set Size vs System Utilization 55

Figure 4.5 Relative Error vs Task System Size 56

Figure 4.6 Testing Set Size vs Task System Size 56

Figure 4.7 Relative Error vs Resource Period 56

Figure 4.8 Testing Set Size vs Resource Period 56

Figure 4.9 Relative Error vs Approximation (k) 57

Figure 4.10 Execution Time vs System Utilization 57

Figure 4.11 Execution Time vs Task System Size 57

Figure 4.12 Execution Time vs Resource Period 57

Figure 5.1 Supply-bound function with ℓ-feasibility region and line segment $\mathcal{L}^i_{t_a}$ 65

Figure 5.2 Relative Error vs System Utilization 90

Figure 5.3 Relative Error vs Workload Size 90

Figure 5.4 Relative Error vs Resource Period . . . 90
Figure 5.5 Relative Error vs Approximation Parameter . . . 90
Figure 5.6 Execution Time vs System Utilization . . . 91
Figure 5.7 Execution Time vs Workload Size . . . 91
Figure 5.8 Execution Time vs Resource Period . . . 92
Figure 5.9 Cumulative request bound function for arbitrary-deadline sporadic tasks . . . 97
Figure 5.10 Illustration of AFPSINTERSTAGE and triangular region between usbf and sbf. . . . 103
Figure 5.11 Determining number of active jobs in $(t_{a-1}, t_a]$ interval. . . . 105
Figure 5.12 Resource Capacity vs Utilization . . . 108
Figure 5.13 Execution Time vs Utilization . . . 108
Figure 5.14 Relative Error Vs Approximation . . . 108

Figure 6.1 Illustration of exact admission control algorithm. . . . 120
Figure 6.2 Naive reduction in intervals might cause violation of dbi. . . . 122
Figure 6.3 Approximating Y-axes of dbi. . . . 124
Figure 6.4 Approximation point $\hat{P}$ in the $(1+\epsilon)$-region $\mathcal{A}^i$. . . . 125
Figure 6.5 Deriving Upper Bound for Periodic dbi. . . . 135
Figure 6.6 State transition diagram. . . . 139
Figure 6.7 Execution Time vs Arrival Time of Jobs . . . 144
Figure 6.8 Execution Time vs Accepted Jobs . . . 144
Figure 6.9 Accepted Jobs vs Arrival Time of Jobs . . . 145
Figure 6.10 EXACTAC is insufficient for arbitrary aperiodic jobs. . . . 146
Figure 6.11 Illustration of UPDATE-A$(\delta_x, \delta_y, j_k)$ operation on $\mathbb{L}$ . . . 150

Figure 7.1 Space Filling Curves . . . 162
Figure 7.2 Simulation results comparing space complexity of our approach . . . 164

LIST OF ALGORITHMS

Algorithm 1 Pseudo-code for determining minimum capacity for a periodic resource using EDF scheduling algorithm 38

Algorithm 2 Pseudo-code for determining exact fixed-priority schedulability of a component with periodic resource 61

Algorithm 3 Pseudo-code for determining minimum capacity for a periodic resource Ω using fixed-priority scheduling algorithm. 68

Algorithm 4 Pseudo-code for determining minimum capacity for a periodic resource with arbitrary-deadline tasks using fixed-priority scheduling algorithm. 99

Algorithm 5 Pseudo-code for determining number of schedulable jobs within consecutive testing set points 100

Algorithm 6 Pseudo-code for admission control of MAD jobs where interface is a single-step demand-curve. 113

Algorithm 7 Pseudo-code for admission control of aperiodic jobs where interface is a single-step demand-curve. 115

Algorithm 8 Pseudo-code for exact admission control of MAD jobs where interface is an arbitrary demand-curve. 119

Algorithm 9 Pseudo-code for approximate admission control of MAD jobs where interface is an arbitrary demand-curve. 127

Algorithm 10 Pseudo-code for checking upper bound of periodic dbi for admission control of MAD jobs. 136

Algorithm 11 Pseudo-code for approximate admission control of arbitrary aperiodic jobs where interface is an arbitrary demand-curve. 147

LIST OF NOTATIONS

C	Component	8
W	Workload of Component C	4
I	Real-time Interface of Component C	4
τ	Sporadic Task System	30
τ_i	Sporadic Task	26
$\tau_{i,j}$	j-th job of task τ_i	26
e_i	Worst-case Execution Requirement of τ_i	26
d_i	Relative Deadline of τ_i	26
p_i	Period of τ_i	26
o_i	Release Offset of τ_i	154
u_i	Utilization of τ_i	26
U_τ	System Utilization for τ	26
H_τ	lcm of task periods of τ	155
n	Number of Tasks in τ	26
J	Set of Aperiodic Jobs	25
j_i	Aperiodic Job	25
A_i	Arrival Time of j_i	25
E_i	Worst-case Execution Requirement of j_i	25
D_i	Relative Deadline of j_i	25
$\bar{d}_i$	Absolute Deadline of τ_i or j_i	25
N	Number of active jobs in the system	25
$\mathsf{dbf}(\tau_i, t)$	Demand-Bound Function for τ_i	27
$\widetilde{\mathsf{dbf}}(\tau_i, t, k)$	Approximate Demand-Bound Function for τ_i	36
$\mathsf{DBF}(\tau, t)$	Demand-Bound Function for τ	27
$\widetilde{\mathsf{DBF}}(\tau, t, k)$	Approximate Demand-Bound Function for τ	36
$\mathsf{rbf}(\tau_i, t)$	Request-Bound Function for τ_i	29
$\widetilde{\mathsf{rbf}}(\tau_i, t)$	Approximate Request-Bound Function for τ_i	29

$\mathsf{RBF}(\tau_i, t)$	Cumulative Request-Bound Function for τ_i	29
$\widetilde{\mathsf{RBF}}(\tau_i, t)$	Approximate Cumulative Request-Bound Function for τ_i	29
$\mathsf{TS}(\tau)$	Exact Testing Set for EDF-Scheduled Task System τ	28
$\widetilde{\mathsf{TS}}(\tau, k)$	Approximate Testing Set for EDF-Scheduled Task System τ	36
TS_i	Exact Testing Set for FP-Scheduled Task τ_i	28
$\widetilde{\mathsf{TS}}_i(k)$	Approximate Testing Set for FP-Scheduled Task τ_i	36
$\mathsf{demand}(J, T_1, T_2)$	Demand of Aperiodic Jobs in J within Time Interval $[T_1, T_2]$	27
Ω	EDP Resource	8
Π	Period of Resource Model	8
Θ	Capacity of Resource Model	8
Δ	Deadline of Resource Model	8
Θ^*	Exact Capacity Required to Schedule a Component	31
$\widehat{\Theta}$	Approximate Capacity Required to Schedule a Component	9
$\mathsf{sbf}(\Omega, t)$	Supply-Bound Function for Ω	31
Λ	Arbitrary Demand-Curve Interface	32
$\mathsf{dbi}(\Lambda, t)$	Demand-Bound Function for Λ	32
ϵ	Approximation Parameter	7
k	Equivalent to $\lceil \frac{1}{\epsilon} \rceil$	36
$\mathcal{S}_\tau$	System Slack for EDF-Scheduled Task System τ	157
t	Represents an interval of length t time units	26
T	Represents an absolute time instant	26

CHAPTER 1: INTRODUCTION

Real-time and embedded systems have become increasingly complex due to the advent of high-performance microprocessor technology. A modern day automotive system provides numerous functionalities such as adaptive cruise control, collision avoidance, lane-departure warning systems, satellite navigation applications etc. In traditional models, separate Electronic Control Units (ECUs) supporting a single hard real-time application are used. A recent trend is integrating functionalities into a smaller number of more powerful microprocessors. Further, independently-developed applications from different vendors might be accommodated to same processing platform. The motivation for such integration comes from the reduction in size, weight and power (SWaP) of the system as a whole, and at the same time this provides the opportunity to enhance functionality. The design goal of such systems raises issues like resource allocation and partitioning among the applications, and thus, presents a great research opportunity for *compositional real-time* analysis. In this thesis our goal is to obtain efficient solutions for such systems. We begin with describing basic real-time concepts before introducing compositional concepts.

A *real-time system* is a system whose correctness depends on the logical result of computation along with the "timeliness" of completion. In such systems, the main performance metric is the timeliness (e.g., deadline constraints) as predictability of the system is more important than performance. Such systems are usually modeled as set of *jobs* where each job is a sequential set of instructions. Each of the jobs is characterized by its *execution (resource) requirement*, *release time* at which the job is ready to execute and a *deadline*, by which the job must complete its execution. There are two types of real-time jobs: *hard real-time* (HRT) jobs and *soft real-time* (SRT) jobs. For the former, the deadline constraint is rigid, all jobs in the system must execute before its deadline; if a job misses its deadline then the system is incorrect (e.g., a nuclear power plant is an example of a HRT system). For the latter case, the system allows deadline misses for some jobs (e.g., multimedia applications are often characterized as SRT systems). Often a real-time application is modeled by set of periodic or sporadic *tasks* each of which generates a potentially infinite sequence of real-time jobs, with successive job-arrivals separated by a *period*.

Scheduling is the process of assigning a resource (processor, memory etc.) to the applications in the system and deciding the order of their executions. There are two type of scheduling algorithms: *static* and *dynamic*. In this thesis, we will consider only dynamic algorithms in which tasks are scheduled based

on their *priorities*. These algorithms can also be of two types: *fixed priority* (FP) in which priority of tasks are predefined and do not vary at runtime (e.g., rate/deadline monotonic [66]), and *dynamic priority* in which task priority may vary at runtime (e.g., earliest-deadline first [66]). In rate-monotonic (RM) or deadline-monotonic (DM) scheduling algorithms, static task priorities are assigned based on task periods (RM) or deadlines (DM). In earliest-deadline-first (EDF) scheduling algorithm, at any instant of time the scheduler chooses to execute the job of a task which has earliest deadline in time.

A processing resource in real-time systems can be categorized as *uniprocessor* platform or *multiprocessor* platform. In a uniprocessor platform all the execution requirements are satisfied using one processor, whereas in multiprocessor or multicore platform it is assumed that the processing resource is supplied by m processors or cores. The processing resource provided to a real-time application can be either *dedicated* or *partitioned*. In the former case the application gets access to the resource persistently, and in the later case it has to time-share the resource with other applications in the system. A real-time system is *schedulable* if a given scheduling algorithm (e.g., EDF) can schedule the system meeting all task deadlines. A *schedulability test* is the process of determining the schedulability of a system. This test can either be exact or sufficient. In the former case the test correctly determines the schedulability of a given system using a specific algorithm. In the latter case, if the test results schedulable then the system is in fact schedulable; if the test returns not schedulable then we can not positively conclude the schedulability of the system. In the next section, we describe necessary concepts related to compositional real-time systems.

1.1 Compositional Real-Time System

Recent real-time and embedded systems research has increasingly trended towards *open environments* [34] due to the ease of portability and integration of applications in a single shared platform. In such systems, independently-developed real-time and non-real-time applications may coexist and may join and leave the system dynamically. These requirements have inspired compositional real-time research with *hierarchical scheduling frameworks*. Over the past years, component-based design for real-time systems has received considerable attention, as numerous compositional frameworks have been proposed for both uniprocessor and multiprocessor platforms. Such systems must satisfy following properties [81]: (1) *independence:* the schedulability of each component is analyzed independent of any other components; (2) *separation:* in compositional design, a parent component is separated from the child component and interact with each

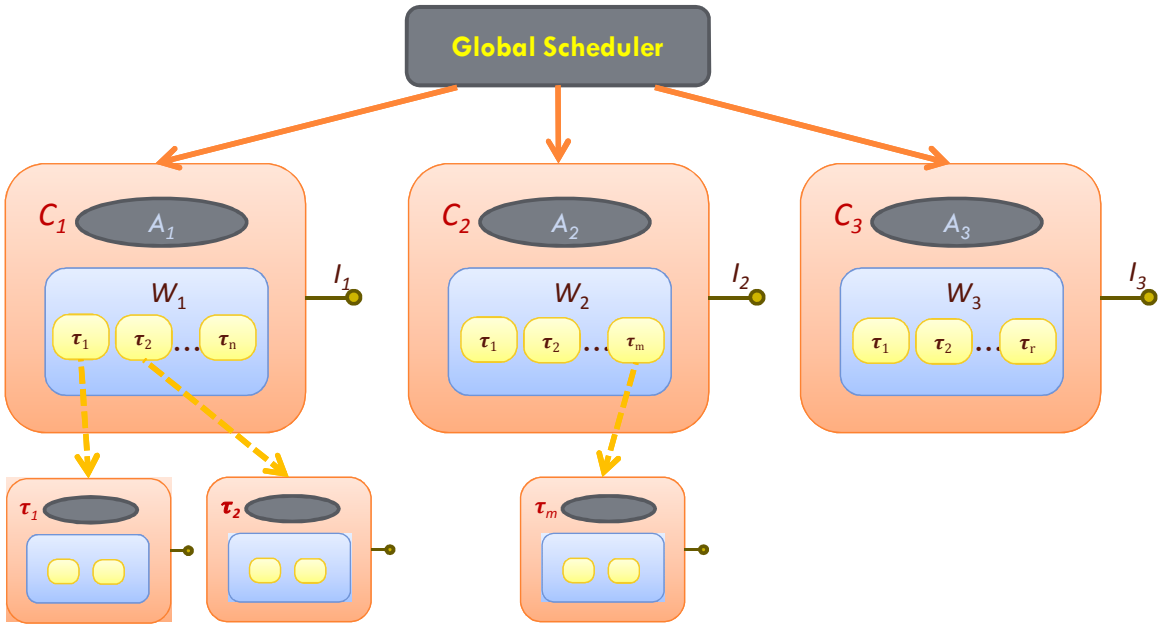


Figure 1.1: Compositional real-time system consisting of three components C_1, C_2 and C_3. The global scheduler allocates the resource among the components, and the local schedulers A_1, A_2, A_3 allocate resource within the respective components. Component C_1 has sub-components τ_1 and τ_2, which forms a hierarchy.

other only through a common interface; (3) *universality:* the component-level or system-level scheduling algorithm is independent, and any scheduling algorithm can be used; and (4) *compositionality:* a parent scheduling model is computed from its child scheduling models such that the timing guarantee of the parent scheduling model is satisfied, if and only if, the timing guarantees of its child scheduling models are satisfied together in the framework. In the next few sections, we elaborate the compositional concepts.

1.1.1 Component-Based Design

As the name suggests, in the design paradigm of a compositional real-time system, a large, complex system is decomposed into smaller and simpler *components*, and developed independently. A major benefit of such design is that the components hide their internal complexity and details from the designer of other components by *component abstraction* and only expose information necessary to use the component. Most proposed frameworks facilitate the abstraction of each component's real-time requirements by specifying a *real-time interface* for each component. While adding the component to a system, only the interface of the component is used to verify the resource requirements of the component.

Let a component C be characterize by the tuple (W, A, I) [83] where W represents the real-time workload generated by C, A represents the component-level scheduling algorithm[1], and I represents the real-time interface for C. We say the component C is *schedulable* upon interface I, if, for any resource supply R that exceeds or equals the resource supply specified by I, W can meet all deadlines when scheduled by A upon R.

In Figure 1.1, a compositional real-time system is shown where the system consists of three independent components C_1, C_2, C_3. Each of the components is independent in a sense that their corresponding workload or component-level (local) scheduling algorithm does not depend on other components in the system. The interface specification for each component represents the timing constraints of the component as a whole, and ensures temporal isolation (i.e., separation) for the components, that is if the resource requirement of one component exceeds its interface specification, the system will ensure that the other components will not be affected by this. A *global scheduler* allocates the resource to the components, and it is independent of local schedulers (i.e., universality). Finally, the system is schedulable if both global and local schedulers can successfully allocate resources among the components (i.e., compositionality) for a specific resource platform.

1.1.2 Hierarchical Scheduling

In compositional real-time system design, the independently-developed components are composed to obtain the original system. Thus, the overall verification of temporal constraints of the system requires only the interfaces of the components in the system. To support component-based design, a multi-level scheduling framework is realized. At the top level of the hierarchy, the global scheduler allocates the resource to heterogenous components (applications). In the next level, a local application-level scheduler schedules tasks (jobs) of the application during the time allocated by top-level scheduler (Figure 1.1).

1.1.3 Real-Time Interfaces

To hide internal complexity from the entire system, components abstract their temporal requirements via interfaces. A real-time interface describes the processing-resource supply that a component requires to meet all real-time constraints. Potential implementations of an interface are via *server-based model* or

[1]Note the component-level scheduling algorithm may differ from the system-level scheduling algorithm. Furthermore, many compositional frameworks permit components to be composed of sub-components.

demand-based interfaces. In the former model, access to the physical resource is provided by the servers, ensuring strong temporal isolation properties (e.g., one component is isolated from the potentially faulty temporal behavior of another component). In the later model, the component demand is precisely modeled by a demand-curve function.

Server-Based Interface

In this model, the component interface is specified by resource partitions; each component receives the processing units for a specific portion of time according to its interface. Given a component with workload, and a local scheduling algorithm, the interface must ensure that all the tasks within the component meet their deadlines. In this model, a component can be assumed to execute within a "server" which allocates the resource partitions based on the component interface, and guarantees temporal isolation among components by ensuring that no component executes beyond its specified interface. Further, given the components and their interface requirements, the system must be able to compose the interfaces such that all timing requirements are satisfied globally.

Demand-Based Interface

In this model, the component interface is generated by demand functions which characterize the maximum amount of processing that the workload of a component will require for any interval of time. The demand interface precisely represents the resource requirement of a component and it is generated using fine-grained techniques such as standard real-time calculus (RTC) [28, 88]. The interface represents the usage and compatibility of a component, and the system designer uses this information to compose the component to the system satisfying all requested real-time constraints. The precise characterization of the interface makes it complex and arbitrary; as a result achieving temporal isolation among components become difficult when components are composed to a shared processing platform.

In Figure 1.2, a compositional system is shown with two components C_i and C_j, where C_i is modeled by server-based interface and C_j is modeled by demand-curve interface. For C_i, the component workload and local scheduling algorithm is known to the designer and by *interface selection*, the resource requirement is determined for this specific component setting. The server ensures that C_i gets its share of resource from the global scheduler (bottom-up approach). In contrast, component C_j is served by a demand-curve

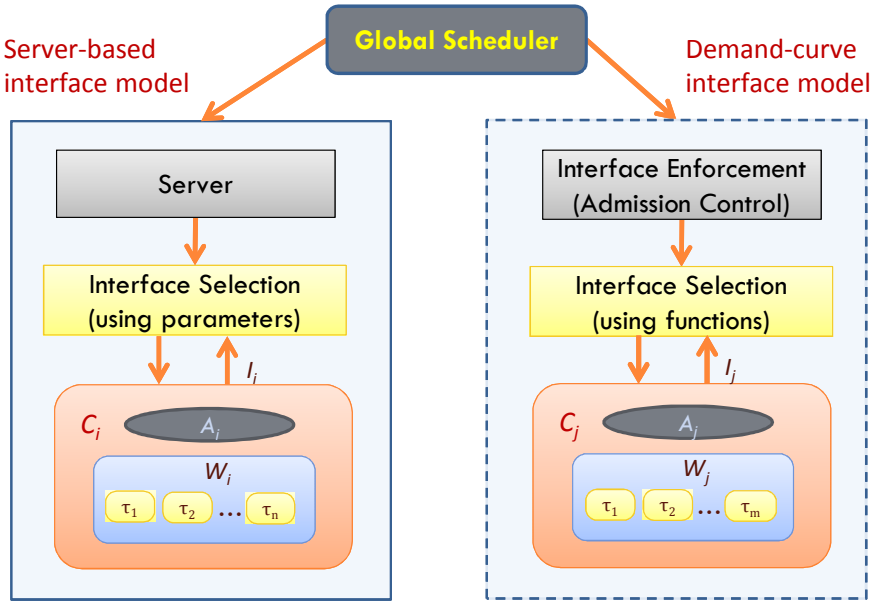


Figure 1.2: Server-based (left) and interface-based (right) resource allocation model for a compositional real-time system.

interface model, where only the interface is known to the system designer. In this model, the component can be compared to a black-box (e.g., a module developed by a vendor who does not want to disclose the internal details other than the component interface), and the workload to the component must be "policed" so that the interface is enforced for the component (top-down approach). An admission controller in this regard performs the *interface enforcement* for the component.

1.1.4 Compositional Schedulability Analysis

In traditional uniprocessor real-time systems, schedulability tests are used to guarantee that the timing constraints of each of the tasks in the system are satisfied. In contrast, for compositional real-time systems, the overall system schedulability is determined in multiple steps. At first, for each component in the system, the component-level schedulability is determined for the component workload and local scheduling scheme. From the resource requirement of the component workload, the component interface is abstracted for each of the components in the system. Finally, using the abstractions, global schedulability analysis is performed to ensure "compositionality" of the system as a whole.

Exact schedulability tests rely on necessary and sufficient schedulability conditions for a system. For a uniprocessor system scheduled by EDF or FP scheduling algorithm upon a dedicated processor, exact

tests require exponential or pseudo-polynomial time complexity. Sufficient schedulability tests for such systems are mostly utilization based and require constant or linear time complexity. Although these tests are computationally efficient, they significantly over-allocate resources to applications and induce lower system utilization. A third possible approach is approximate schedulability tests with a pre-specified approximation parameter. An algorithm $\mathcal{A}$ is called *fully polynomial time approximation scheme* (FPTAS) for a minimization problem $\mathcal{P}$, if given $\epsilon > 0$, it solves $\mathcal{P}$ in time polynomial to input size of $\mathcal{P}$ and $\frac{1}{\epsilon}$. If S^* represent optimal solution to $\mathcal{P}$ and $\widehat{S}$ represent solution returned by $\mathcal{A}$, then it must be that $\widehat{S} \leq (1+\epsilon)S^*$. The $(1+\epsilon)$ factor is called the *approximation ratio* of the produced solution.

Sufficient solutions may be impractical for developing real-time systems in which resources are very scarce. In real-time open systems where applications may be added or removed dynamically, it is important to provision resources efficiently at run-time and an efficient allocation algorithm is desirable. A parametric approximate algorithm allows system designer to pre-specify an arbitrary level of accuracy (ϵ) in obtaining a solution with a quantifiable trade-off with the computation efficiency of obtaining the solution. In this thesis our goal is designing computationally efficient algorithms on real-time guarantee verification, as well as providing the system designer control over accuracy of resource allocation.

1.2 Fundamental Problems in Compositional Systems

In the context of compositional real-time systems, we address three problems in this thesis: *allocation*, *enforcement* and *slack reclamation*. We address efficient allocation of processing resource among the components of the compositional system (Section 1.2.1) . We provide efficient techniques to ensure that the interface of each of the components are strictly enforced (Section 1.2.2). While enforcing a given interface, we devise a method to reclaim slack of processing resource at runtime to allocate possible overruns (Section 1.2.3). In the next section, we give an overview of the allocation problem we addressed in obtaining approximate schedulability tests for compositional systems with EDF- or FP-scheduled components.

1.2.1 Allocation: Minimization of Interface Bandwidth

Although the various proposed compositional frameworks differ in the component information that is exposed to the system by real-time interfaces, a common necessary attribute is the *interface bandwidth*. The interface bandwidth simultaneously quantifies the fraction of the total system resource supply that a

component C will require to meet its real-time constraints and the component C's "interference" on the resource supply provided to other system components. Thus, an important fundamental problem in design and analysis of compositional real-time systems is the *minimization of real-time interface bandwidth* (MIB-RT).

Let $I \in \mathcal{I}$ denote a real-time interface I for component C from the set $\mathcal{I}$ of all possible real-time interfaces in a given compositional framework. Let $\beta(I)$ denote the interface bandwidth of real-time interface I. We say the component C is schedulable upon interface I, if, for any resource supply R that exceeds or equals the resource supply specified by I, component workload W can meet all deadlines when scheduled by algorithm A upon R. With these notations, our goal is to minimize interface bandwidth $\beta(I)$ for a specific compositional framework: the *explicit deadline periodic* (EDP) [39] resource model.

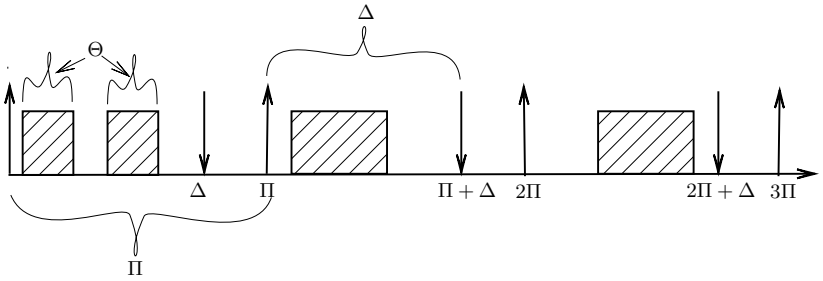


Figure 1.3: An example of resource allocation in the EDP model.

In the EDP model (Figure 1.3), a resource Ω is characterized by a three-tuple (Π, Θ, Δ). The interpretation of such a resource is that a component C executed upon Ω is guaranteed Θ units of processing resource supply for successive Π-length intervals (given some initial starting time). Furthermore, the Θ units of resource supply must be provided within Δ ($\leq \Pi$) time units after the start of the Π-length interval. For $\Omega = (\Pi, \Theta, \Delta)$, Π is referred to as the *period of repetition*, Θ is the *capacity*, and Δ is the *relative deadline*. The EDP model generalized the simpler *periodic resource* model [81] which represented a resource by (Π, Θ) where relative deadline of the resource is implicit (i.e., $\Delta = \Pi$). The advantage of the EDP model over the periodic resource model is to permit tighter control of the resource allocation by reducing the *starvation period* or *no-supply period* for the resource model. The starvation period is the maximum amount of time at which a component using any resource model will not receive any resource from the system. The EDP model provides the flexibility of reducing starvation period by tuning Δ. Moreover, the resource model provides optimal resource to a component when $\Delta = \Theta$ [38, 39]. The results obtained in this thesis apply to both EDP model and the simpler periodic resource model.

The MIB-RT problem for the EDP model can be stated as follows: *for any component C determine capacity $\widehat{\Theta}$, period $\widehat{\Pi}$ and deadline $\widehat{\Delta}$ such that interface bandwidth*[2] $\frac{\widehat{\Theta}}{\widehat{\Pi}}$ *is minimized and C is schedulable by $\mathcal{A}$ upon a resource $\Omega = (\widehat{\Pi}, \widehat{\Theta}, \widehat{\Delta})$.* Therefore, to solve MIB-RT, we need to address two subproblems: capacity determination and period selection. Previous techniques are known for determining $\widehat{\Pi}$ and $\widehat{\Delta}$ [38,44]. Given a capacity determination algorithm $\mathcal{A}$, and a range of values of period Π_{lower} to Π_{upper}, the period selection algorithm selects period Π and uses $\mathcal{A}$ to determine capacity Θ such that the bandwidth Θ/Π is minimized (Figure 1.4). Thus, given those methods, we narrow the scope of MIB-RT as follows: *for any component C, given Π and Δ, determine a capacity $\widehat{\Theta}$ such that $\frac{\widehat{\Theta}}{\Pi}$ is minimized and C is schedulable by $\mathcal{A}$ upon a resource $\Omega = (\Pi, \widehat{\Theta}, \Delta)$.* We solved MIB-RT where component-level scheduler $\mathcal{A}$ is EDF or FP.

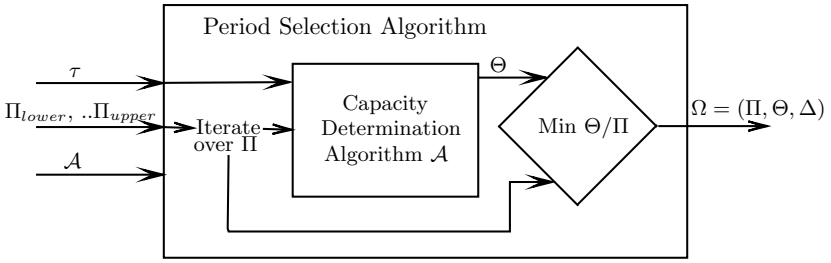


Figure 1.4: MIB-RT problem for periodic resources

The MIB-RT problem has been extensively studied for the various real-time compositional frameworks. The currently-known solutions to MIB-RT generally fall into one of two categories: *exact* or *over-provisioned*. Exact solutions to MIB-RT (e.g., see Easwaran et al. [39] or Lipari et al. [62,65]), determine the exact minimum bandwidth necessary to meet the component's real-time constraints under a given framework – i.e., if the bandwidth provided by the system is less than the exact minimum bandwidth for component C, some real-time constraint will be violated for C. Typically, exact solutions for MIB-RT rely upon modifications to exact schedulability techniques for non-compositional real-time systems [20, 56] which are computationally expensive. Furthermore, MIB-RT with $\mathcal{A}$ equal EDF is clearly co-NP-hard by reduction to uniprocessor EDF schedulability (which was recently shown to be co-NP-Hard by Eisenbrand and Rothvoß [42]); a task system is EDF-schedulable if and only if the task system can be scheduled on a periodic resource $\Omega = (\Pi, \Theta, \Pi)$ with $\Theta \leq \Pi$. Thus, it is unlikely that exact polynomial-time algorithms

[2]We abuse terminology slightly and allow (Π, Θ, Δ) describe both the resource Ω and an interface for C.

exist for MIB-RT. Over-provisioned solutions to MIB-RT (e.g., [5, 39]) for a component C also allocate sufficient bandwidth to C to ensure that all real-time constraints are met; however, the allocated bandwidth may be more than necessary – leading to a potential waste of processing resources. Advantageously, an over-provisioned solution is typically computationally much less expensive than an exact solution since its running time is linear or polynomial in the representation of C's real-time constraints.

The speed-of-analysis for MIB-RT is often a critical issue in compositional real-time systems where components may dynamically join the system, since efficient admission tests based on interface bandwidth are required to quickly determine whether a component may be admitted. To address this need and the current gap for MIB-RT between computationally-expensive exact solutions and computationally-inexpensive over-provisioned solutions, we propose investigation into a third category of solution for MIB-RT: *parametric approximate solutions*. A parametric approximate solution allows the component designer to prespecify an arbitrary level of accuracy in obtaining a solution to MIB-RT; however, the desired level of accuracy has a quantifiable trade-off with the efficiency of obtaining the solution. In other words, an increased level of accuracy requires an increased amount of computation.

Furthermore, we require that the complexity of the algorithm to obtain the approximate solution is polynomial both in terms of $\frac{1}{\epsilon}$ and in the representation of C. In other words, one of our goals is to obtain a fully-polynomial-time approximation schemes (FPTAS) [92] for the MIB-RT problem under various compositional real-time frameworks.

1.2.2 Enforcement: Admission Control

In the previous section, we addressed efficient interface (bandwidth) determination for server-based interface models, where the model enforces the interface via servers (Section 2.1.1). In this section, we address interface enforcement for demand-based or demand-curve interfaces. Consider hard real-time applications developed by different vendors with the real-time requirements specified by their interfaces. Compositional analysis must be performed to integrate such applications into a common hardware platform in a real-time open environment. The system designer must allocate resources for each of the components (applications) such that the performance is guaranteed as well as temporal isolation among the components are strictly enforced (e.g., one component is isolated from potentially faulty temporal behavior of another component). A popular approach for achieving temporal isolation between components is to use

server-based interfaces; each component may execute within a system-provided server that allocates the resource based upon the component's real-time interface. The main disadvantage of this approach is that it requires some over-provisioning of resources to guarantee temporal isolation [8]. A more precise alternative to server-based approach is to use a *demand-curve interface* to specify the amount of computational resources (over any interval of time) that the components will require. The well known real-time calculus (RTC) frameworks [28, 88] are examples of this approach. However, strict temporal isolation is currently difficult to achieve between components specified by demand-curve interfaces, as there is no known online "policing" protocol for ensuring that a system does not violate an arbitrarily-specified demand-curve interface [17].

In this thesis, we address the lack of interface-policing protocols for the more precise demand-curve interface models. To achieve this, we propose admission controllers for components comprising aperiodic hard-real-time jobs. These admission controllers will ensure temporal isolation by checking whether a newly-arrived job may be admitted for execution in a component without violating its demand-curve interface. Furthermore, we describe the implementation of an associated server which can enforce (at runtime) the temporal isolation once a job is admitted to the system. Our primary design goal is the development of admission controllers that are both theoretically and practically efficient; i.e., we can prove tight, polynomial bounds on computation complexity and observe low overhead in an implementation.

1.2.3 Slack Reclamation

In traditional hard real-time systems, schedulability analysis is often performed offline with the assumption of worst-case workload demand. The analysis guarantees that the system will not violate any deadline given that it behaves predictably. However, the runtime behavior of the system may vary due to various factors such as resource availability, overrun, other runtime dependencies (e.g., memory, I/O, cache, etc.). Due to such variability, worst-case execution time (WCET) analysis often gives pessimistic bounds on the execution time of real-time jobs that are rarely achieved. Thus, at runtime, there is often an opportunity to make use of the processing time *slack* left by the jobs that did not execute up to their WCET.

In the last part of the thesis, we address the challenge of determining "allowable" slack for any job in a system (or component) while still maintaining system schedulablity. We provide a novel technique to efficiently compute system slack at runtime, which can be used to *allow* a job to *overrun* if there is enough

slack in the system. For this problem we assume a compositional system where components may have arbitrary interface (i.e., server-based or demand-based).

1.3 Applications

Real-time application domains include automotive, automation, avionics, consumer electronics, industrial process controller and medical systems among others. Compositional system design has wide applicability in such systems. In the next few subsections, we describe potential application domains where our theoretical contribution in this thesis may be applied.

1.3.1 Automotive

In automotive electronics, the advent of high-performance embedded microprocessors has made possible complex functionalities in different domains such as active safety (e.g., emergency brake assist system), driver assistance (e.g., cruise control, blind spot detection, park assist system etc.), passenger comfort (e.g., automatic climate control), infotainment systems (e.g., navigation system) etc. With the growing demand of functionality, the number of Electronic Control Units (ECU) has also increased in modern-day vehicles (e.g., Volkswagen Phaeton 2002 has about 70 to 100 ECUs), which raises issues like increase in copper wiring, cost, space and power of the vehicle. Instead of using sperate processing platform (i.e., ECU) per application, multiple applications can be combined into same ECU to take the benefit of powerful microprocessors. The design goal of integrating a number of real-time applications onto a single processing platform necessitates compositional analysis to ensure real-time guarantee for the system.

The well known open framework *AUtomotive Software ARchitecture* (AUTOSAR) [10] can be used in this regard to implement component-based systems. This standardized architecture for automotive system modularizes the softwares (applications) into components with well-defined interfaces, and separates hardware and software in the subsystem. Using this, complex applications can be decomposed into simpler ones, developed independently, and their resource and temporal requirements can be represented by interfaces. Finally, the timing constraints of the entire system will be guaranteed by compositional analysis by integrating multiple real-time applications into a shared hardware platform (ECU).

1.3.2 Avionics

Integrated Modular Avionics (IMA) represents real-time computer network for airborne systems consisting of computing modules which support numerous applications of differing criticality levels. ARINC 653 (Avionics Application Standard Software Interface) [84, 96] is a software specification for space and time partitioning in safety-critical avionics real-time operating systems. It allows hosting of multiple applications of different software levels on the same hardware in the context of a IMA architecture. In this specification, the real-time system of an aircraft consists of one or more core modules which are connected using switched Ethernet. The core modules are hardware platforms consisting of one or more processors, and provide space and temporal partitioning for independent execution of avionics applications. Each independent application (also known as partition) has one or more processes representing its real-time resource demand. Therefore, the workload on a single processor in a core module can be described as a two-level hierarchical real-time system. Each application has a (local) application specific scheduler which schedules processor time among the processes (tasks) in that application. All the applications that are allocated to the same processor are then scheduled among themselves using a (global) resource scheduler. Compositional analysis can be applied for ARINC-653 platform, in which the system can be modeled as multi-level hierarchical scheduling [41] system.

In modern-day aircrafts, system design is performed manually through interactions between application vendors and system designers. Using compositional schedulability analysis, this process can be automated [41] and served as a framework for the design of application level schedules. Also the analytical correctness guarantee provided by this method can be used as system certification. The MIB-RT problem we address in this thesis can be applied for this setting to obtain efficient interfaces for the avionic applications. Our approach provides system designer with a choice of accuracy to trade with the speed-of-analysis.

1.3.3 Energy-Aware Systems

We addressed efficient slack reclamation at runtime in the context of compositional real-time systems. Once system slack is computed, it may be utilized in a variety of ways (e.g., permitting a job to overrun its WCET). An important potential application of slack reclamation is in energy-aware systems. In designing such system, the biggest challenge is to minimize energy consumption by lowering the processor

frequency/voltage to reduce power dissipation. However, a real-time system's task deadlines constrain how much the frequency may be reduced. If there is an account of system slack at runtime, it will be easy to determine whether the system can safely execute at a lower frequency state (and thus increase the WCET of executing jobs if required).

1.4 Contribution

The thesis of this dissertaion is as follows:

> *Currently, server-based interfaces ensure strong temporal isolation among components at the cost of resource over-provisioning whereas demand-based interfaces precisely model the resource demand of a component without the guarantee of temporal isolation. For both these models, we show that efficient and effective resource allocation as well as strict temporal isolation among components can be achieved. Specifically, we obtain efficient and near-optimal bandwidth allocation schemes and admission controllers for periodic resource model and arbitrary demand-based interface respectively. Furthermore, efficient slack reclamation technique can be obtained to allocate unused processing resources at runtime while still enforcing the given interface.*

To support the thesis, we make the following contributions in this thesis. For server-based interfaces, we address resource over-provisioning among components by proposing computationally efficient approximate bandwidth allocation algorithms when components are scheduled upon periodic resources. For demand-based interfaces, we address the enforcement of temporal isolation among components by developing admission controllers for components specified by arbitrary demand interfaces.

- We propose a parametric approximation algorithm for bandwidth allocation of periodic resources for earliest-deadline-first scheduled components [46, 47]. Exact bandwidth allocation for MIB-RT of sporadic task system uses either exponential-time or pseudo-polynomial-time techniques, and utilization-based sufficient techniques over-provision bandwidth. A parametric approximation scheme in this case will allow component designer to specify an arbitrary level of accuracy (ϵ) for bandwidth computation, and trade accuracy for computational complexity. Furthermore, the complexity of our algorithm is polynomial both in terms of $\frac{1}{\epsilon}$ and in the representation of C. In other words, we obtain

a fully-polynomial-time approximation schemes (FPTAS) [92] for the MIB-RT problem. We also show, via simulation, that our algorithm is quite accurate over synthetically generated task systems even for medium-sized values of ϵ.

- For fixed-priority scheduled components of server-based model, we propose a parametric approximation algorithm for bandwidth allocation of periodic resources where task deadlines are less or equal task periods [35]. The complexity of our algorithm is polynomial in $\frac{1}{\epsilon}$ and in the representation of C, and the runtime is significantly smaller compared to the exact algorithm.

- For demand-based interface models, we address the interface enforcement problem to ensure strict temporal isolation among components by proposing exact admission control algorithm for aperiodic workload [36]. Given a simple demand interface, our algorithm runs in constant time for *monotonic absolute deadline* (MAD) jobs[3] and $O(\log N)$ time for arbitrary aperiodic jobs where N is the number of active jobs in the system at any time.

- For arbitrary demand-curve interfaces, we propose exact admission control scheme for aperiodic workload and show that it is computationally infeasible for long-running online system. For this setting we propose a parametric approximate admission control algorithm, which has polynomial time complexity in terms of number of active jobs in the system N and the approximation parameter ϵ [37]. We implement each of our proposed admission controllers and show that our approximate admission controller is both efficient and precise (in comparison to the exact) via simulation.

- To address efficient slack reclamation, we characterize the optimal runtime system slack at any discrete time instant with the guarantee that all future jobs in the system will remain schedulable. We describe a straightforward slack determination mechanism which is of exponential space complexity. To efficiently store and access the data, we investigate the use of space-filling curves along with advance tree data structure. Furthermore, we propose a simple approximation on the stored slack data to further reduce the space complexity of the problem. Simulation results comparing the proposed approaches (exact and approximate) are presented which shows significant improvement in space complexity.

[3] See Section 3.1.1 for illustration of MAD aperiodic jobs.

1.5 Organization

The remainder of the thesis is organized as follows. We review prior related research on various compositional real-time frameworks in Chapter 2. We give necessary background and notations in Chapter 3 (see List of Notations for a complete list of notations used in the report). In the next two chapters (Chapter 4 and 5) we address the MIB-RT problem in the context of compositional frameworks executing upon explicit-deadline periodic resource. In Chapter 4, we present our fully-polynomial-time approximation algorithm for EDF scheduled components and prove its correctness and approximation ratio. We extend the results for fixed-priority component level scheduling algorithm and give an FPTAS in Chapter 5. In Chapter 6, we propose exact and approximate admission control algorithm for simple demand interfaces and arbitrary demand interfaces respectively. We address runtime slack determination techniques for a compositional system in Chapter 7. Finally, in Chapter 8 we finish the thesis with the summary of contributions and future research directions.

CHAPTER 2: RELATED WORK

Over the past decade, numerous compositional real-time frameworks have been proposed, since the original seminal works introducing *real-time open environments* by Deng and Liu [34] and *resource kernels* by Rajkumar et al. [74]. In this chapter we briefly survey current literature on compositional frameworks for both uniprocessor (Section 2.1) and multiprocessor platforms (Section 2.2). We explore server-based models and interface-based models in Section 2.1.1 and 2.1.2 respectively, and highlight key differences of these models with our contribution in this thesis.

2.1 Uniprocessor Platform

Research in compositional schedulability analysis for uniprocessor platforms has matured over years. For the two-level hierarchical framework of Deng and Liu [34], Kuo and Li [54] developed exact schedulabiltiy conditions for rate-monotonic (RM) system-level scheduler with harmonic task periods. Lipari and Baruah gave exact conditions [62, 65] for dynamic priority (EDF) system-level scheduler for this model. The common assumption for these initial works was that the system-level scheduler has full processor capacity (100% utilization), which limits the hierarchical framework to two levels only. A more general approach was proposed by Regehr and Stankovic [75] in which they used real-time guarantee conversion between parent and child components. In their model, the child component is schedulable if the guarantee provided from parent component can be converted to the guarantee that the child component demands.

To support multi-level hierarchical frameworks in uniprocessor platforms, two types of compositional analysis techniques are used: a) resource model (server) based analysis and b) demand-bound function based analysis. In the next two sections we describe different frameworks of these models.

2.1.1 Server-Based Models

In resource model based techniques, a component workload is abstracted by partial resource supply (i.e., processor execution). In other words, resource models can be used to specify the real-time guarantees that a parent component provides to its children and can be assumed as "servers". Compositional analysis techniques have been proposed for *bounded-delay resource model* [43,69,82], *periodic resource model* [5,

31, 63, 76, 81] and *explicit deadline periodic* [38, 39] (EDP) model; we will cover these models in more details in this section.

Real-Time Open Environment

The concept of an *open environment* for real-time systems with uniprocessor platforms was first proposed by Deng and Liu [34], where real-time and non-real-time applications may coexist in a system and may join and leave the system dynamically. This allows admission of new real-time tasks to be independent of the existing tasks in the system and thus removes the requirement of global schedulability analysis whenever a new task arrives in the system. The real-time applications supported by this model may have non-preemptive tasks, aperiodic or sporadic tasks (i.e., tasks having release time jitter) and may be scheduled by priority driven algorithms. The concept was first discussed for EDF kernel scheduler [34], and was later extended to fixed-priority kernel scheduler [55].

Resource Kernel

In [74] Rajkumar et al. proposed the concept of *resource kernels* to provide timely, guaranteed and protected use of OS resource for concurrently running real-time and multimedia applications with different timing constraints. In their model, applications specify their resource demands and the kernel satisfies the demands using hidden resource management, and thus provides a separation of concern. This allows OS-subsystem-specific customization (e.g., independent scheduling algorithms, extension) and simultaneous access to multiple resources by resource decoupling and thus resolving critical resource dependencies immediately.

Bounded-Delay Model

Feng and Mok [43] proposed the concept of *temporal resource partitions* where processing resource is available to each application at specific time partitions. This allows multiple application to share the same resource platform and thus support hierarchical resource sharing. However, the characterization of partitions can become too specific to the applications in the system which makes this model less flexible. To overcome this, they introduced more general *bounded-delay resource model* [69] to implement open system environments. In this model, a resource partition is specified by a tuple (α, Δ) where $0 \leq \alpha \leq 1$

is the availability factor of the partition and Δ is the maximum delay of the time intervals of any length in the partition. Intuitively, each partition receives a fraction α of the total amount of processor capacity with partition delay at most Δ. This model achieves a clean separation between a parent component and its children, and has been used for compositional schedulability analysis [43, 82]. Henzinger and Matic [48] have also extended these techniques to support compositional analysis. Almeida and Pedreiras [5] developed sufficient, polynomial-time bandwidth allocation techniques for fixed-priority scheduling upon temporal partitions.

Periodic Resource Model

The *periodic resource model* (or periodic servers) has been introduced by Saewong et al. [76], Lipari and Bini [63], and Shin and Lee [81]. This model characterizes resource supply guaranteed to any component in compositional system (along with algorithms for MIB-RT described in the introduction) with periodic resource allocation behavior, that is, the real-time guarantee of the parent model is satisfied if and only if the real-time guarantee of the child model is satisfied. A periodic resource model is characterized by the tuple (Π, Θ) where a resource or budget of Θ time units (*capacity*) is provided within a time interval of Π (*period*).

For RM scheduled components upon periodic resources, Saewong et al. [76] introduced an exact schedulability condition based on worst-case response time analysis, and Lipari and Bini [63] developed an exact, pseudo-polynomial-time algorithm for MIB-RT based on time demand analysis. Similarly, for dynamic-priority (EDF) scheduled components, Shin and Lee [81] have presented an exact schedulability condition based on time demand analysis. To analyze hierarchical scheduling frameworks, these studies have also presented composition techniques for the component interfaces generated by the resource models.

Explicit Deadline Periodic Resource

Easwaran et al. [39] generalized the periodic resource model as *explicit deadline periodic* (EDP) model by introducing a deadline parameter to the component interface (see Section 1.2.1). They developed optimal interface generation and composition techniques and gave exact algorithms for MIB-RT for their model.

State-of-the-art algorithms for bandwidth allocation use either exponential-time or pseudo-polynomial-

time techniques for exact allocation, or linear-time, utilization-based techniques which may over-provision bandwidth. In this thesis, we propose research into a third possible approach: parametric approximation algorithms for bandwidth allocation in compositional real-time systems with the goal to allow a component designer to specify an arbitrary level of accuracy for bandwidth computation, and trade increased accuracy for increased computational complexity. We use EDP model for efficient interface determination problem addressed in Chapter 4 and 5.

Fixed-Priority Servers

The schedulability analysis of fixed-priority-scheduled components upon periodic resources can be compared to existing fixed-priority servers [57, 85–87]. When system consists of both periodic and aperiodic jobs, the periodic jobs require fraction of resource proportional to their utilization. A fixed-priority server reserves a fraction of resource for the upcoming aperiodic jobs, and serves them whenever such job arrives in the system depending on the server "budget". In deferrable server [57, 87], the server budget (C) is replenished in periodic manner at the beginning of server periods (T). If there is any remaining budget in a period T, it can be used in the next server period. In sporadic server [86], the initial server budget is C and it is consumed when an aperiodic job arrives. The budget is replenished after T time units from the arrival of an aperiodic job, and it is replenished the amount consumed by the aperiodic job. The aperiodic jobs have highest priority for both these models and if the budget is exhausted while executing a job, the job is suspended unit next replenishment occurs.

For the periodic resource models discussed in the previous sections, the starvation or no-supply period of the resource model can be considered as a special task (similar to the aperiodic server job) with highest priority, and traditional uniprocessor schedulability analysis can be performed to obtain a solution to MIB-RT. Further, response time bounds can be derived similar to [32] to obtain efficiency in traditional fixed-priority scheduling for FP-scheduled components scheduled upon periodic resources (Section 5.2).

To the best of our knowledge, we are currently not aware of any approximation algorithm (prior to our work) with known approximation ratio developed to address the MIB-RT problem for any of the real-time compositional frameworks. However, research by Albers et al. [2] has developed parametric algorithms, without known approximation ratios, for the *hierarchical event stream model*. The goal of our work is to fill this needed gap by obtaining an FPTAS for MIB-RT in the periodic resource model with uniprocessor

platforms.

2.1.2 Interface-Based Models

Recently, researchers have also focused on characterizing component interface by demand functions which describe the maximum amount of processing that the workload of a component requires for any interval of time. The motivation of such design is to analyze essential characteristics of a system at early design stage; before much time is invested in detailed implementation. An example of such characteristics are maximum delay and throughput constraints, on-chip memory requirements, dimensioning of architectural elements etc. In interface-based system design, system components are characterized by component interfaces, which describe the usage and compatibility of the components to the entire system. A component interface provides enough information to decide whether two or more components can work together satisfying all requested real-time constraints. However, achieving temporal isolation in such model is not trivial when components share same processing platform.

Real-Time Calculus

Several interface-based frameworks have been proposed [89, 93, 94] for subsystems[1] of a compositional real-time system. For these frameworks the compositional design is based on Real-Time Interfaces [4] and the analysis is based on Real-Time Calculus (RTC) [28, 88]. These models provision for inherent constraint propagation and address important design issues as expected maximum delay, maximal task activation rate supported, least hardware resource requirement, amount of remaining resource, etc. For example, Thiele et al. [93] proposed the concept of *interface-based design* to compute *demand curves* and *service curves* for interface abstraction of a component in a compositional real-time system. They applied existing interface theory [4, 33] into real-time context by developing a real-time calculus [88] for these interfaces. In their model interfaces for the subsystems are "assumed", and the system "guarantees" the interface to the subsystem. Compositional techniques have also been developed based on *incremental design* of components in the system.

Unlike the server-based resource model, the demand interface model precisely characterizes the subsystem demand. If the system has to guarantee the resource demand accurately to different subsystems,

[1] We use the term "subsystem" as a synonym for the term "component" in compositional setting.

enforcing strict temporal isolation among subsystems becomes difficult to achieve. In Chapter 6, we address this problem by developing admission controllers for simple and arbitrary demand interfaces. In the next few sections, we discuss state-of-art admission controllers for a subsystem consisting of aperiodic and/or periodic jobs.

Bandwidth Sharing Server

Linear-time exact admission tests for scheduling periodic and aperiodic jobs have been proposed in [1, 78], which can be used for admission control. Lipari and Buttazzo [64] proposed *Bandwidth Sharing Server (BSS)* algorithm which provides precise isolation between subsystems. In [62], this model has been extended to support aperiodic servers with different subsystem-level schedulers. Using a similar approach, Andersson and Ekelin [9] proposed an $O(\log N)$ exact admission controller for aperiodic and periodic jobs in a non-compositional setting where N is the set of active jobs in the system. In Section 6.2 we extend their techniques to apply to admission control for simple (single-step) demand-curve interfaces (see Section 3.3.2 for more details).

Demand-Bound Server

A recent paper by Kumar et al. [53] has proposed a Demand-Bound Server (DBS) for scheduling jobs according to a demand-curve interface. The proposed server successfully achieves the goal of providing temporal isolation between subsystems specified precisely by a demand-curve. However, the approach has fundamental differences with our proposed approach in Section 6.3. First, Kumar et al. do not provide an admission controller for DBS; thus, if a subsystem incorrectly generates workload which exceeds the specified demand curve, the over-allocation error would only become apparent when the subsystem misses a deadline. In our approach, we seek to identify jobs that exceed a subsystem's demand-curve interface before they are admitted to the scheduler. This approach permits a subsystem designer to identify and recover from potential over-allocation errors early before a temporal violation has occurred. The second fundamental difference is that Kumar et al. assume that jobs are scheduled in first-come first-serve (FCFS) order. Our general approach makes no assumptions regarding the underlying execution of admitted jobs; we only guarantee that the set of admitted jobs does not violate the demand-curve interface. In Kumar et al. [53], while a general mathematical model is presented for arbitrary demand curves, the server algorithm

has only been specified for a very simple periodic demand function corresponding to the demand of a single periodic task. In Chapter 6, we perform admission control and ensure temporal isolation for an arbitrarily-complex demand curve.

2.1.3 Slack Reclamation

Several techniques exist in the literature to reclaim available unused resources for both uniprocessor and multiprocessor scheduling platforms. System slack or spare capacity determination in the context of fixed-priority scheduling has been explored in the Ph.D. thesis of Davis [30]. Existing server algorithms (e.g. CBS [1], BSS [62], etc.) address overrun and resource reclamation of jobs which are mostly aperiodic in nature. When a system consists of both aperiodic and periodic tasks and aperiodic tasks are scheduled by constant bandwidth server (CBS), Caccamo et al. [27], [72] proposed techniques to reclaim unused bandwidth allocated to the aperiodic tasks. Marzario et al. [67], developed similar techniques to utilize unused bandwidth evenly to the jobs in the system to handle computational overload. Although these techniques efficiently handle system overrun for aperiodic workload, none of the approaches guarantee that the overrun will not violate the schedulability of future periodic/sporadic jobs.

Chetto and Chetto [29] determined system idle-time for a set of periodic tasks scheduled by EDF by computing a slack table containing location and duration of slack over the hyperperiod. Andersson and Ekelin [9] proposed an exact admission controller for system with integrated aperiodic and periodic tasks. Using similar techniques as [29], they obtained slack from the periodic jobs, and used it to accommodate aperiodic jobs and overrun situations. In Chapter 7, we use sporadic task model with EDF, in which case it is not possible to determine slack location and duration in the schedule beforehand. Instead, we determine worst-case available slack for each active job in the system at runtime.

Given a feasible preemptive uniprocessor system, Bertogna and Baruah [22] provide a method to compute non-preemption period for jobs to reduce preemption overhead at runtime. However, their approach is not dynamic in a sense that the computed slack period does not take into account of the execution times of the active jobs. Our goal here is to determine system slack online, so that runtime unpredictability can be better accommodated.

A significant body of work has also been done in the context of slack reclamation in energy-aware systems exploiting the processor's dynamic voltage scaling (DVS) feature. In this case slack is determined

at runtime by changing the processor voltage such that the overall energy consumption is minimized [15, 50, 52, 73, 95]. Unlike DVS, in Chapter 7 we aim to determine runtime system slack for a fixed processor voltage/frequency.

2.2 Multiprocessor Platform

Along with uniprocessor platforms, several compositional frameworks have been proposed in the literature for multiprocessor resource platforms where components are modeled as hard or soft real-time sporadic tasks. Anderson et al. [6] proposed a two-level multiprocessor scheduling framework with Pfair scheduler [7, 19]. Shin et al. [79] extended their periodic resource model [83] for multiprocessor platforms. Leontyev and Anderson [59, 60] proposed hierarchical scheduling framework of soft/hard real-time tasks in multiprocessor platforms. Since most of these frameworks devise sufficient solutions for MIB-RT, we can extend our solution for uniprocessor platforms to obtain approximate solutions with constant-factor approximation ratio for multiprocessor platforms.

For hard real-time applications, Bini et al. [24, 26] proposed compositional frameworks with underlying multiprocessor resource platforms. The proposed *multi-supply function* (msf) abstraction [26] and *parallel supply function* (psf) abstraction [24] represent resource supply to an application from a virtual processor characterized by m physical processors. The msf abstraction represents set of m supply functions[2] each corresponding to supply from a single processor, and the psf abstraction represents set of m supply functions where i-th supply function represent resource supply with atmost i-processors running in parallel. For these abstractions, we can develop admission controller for the applications similar to our approach described in Chapter 6, and thus enforce strong temporal isolation among applications running in multiprocessor platforms.

Please note that, we do not address multiprocessor platform in this thesis.

[2] See Definition 4 of Section 3.3, which characterizes supply function for periodic resources in uniprocessor platforms.

CHAPTER 3: MODEL AND NOTATIONS

In this chapter, we introduce the reader to the background and notation for the task model, workload functions, and interface models that we use throughout the thesis. In Section 3.1, we describe the task models assumed for the problems considered in this report. In Section 3.2 we quantify the workload of the task models, and in Section 3.3, we describe the resource model and interface model of compositional real-time system used in our specific problem settings.

3.1 Task Model

While addressing the resource allocation problem (Chapter 4, 5) and slack reclamation problem (Chapter 7) we have modeled a real-time component as a set of sporadic tasks [70], since this model is more generalized than traditional periodic task model [66]. For enforcing demand interfaces in Chapter 6 we have assume that a component consists of a set of aperiodic jobs.

3.1.1 Aperiodic Job Model

Each aperiodic job j_i is characterized by an *arrival time* A_i, a *worst case execution requirement* E_i, and a *relative deadline* D_i; a job j_i is denoted by the three-tuple (A_i, E_i, D_i). We also denote the *absolute deadline* for j_i as $\bar{d}_i \stackrel{\text{def}}{=} A_i + D_i$. A job set $J = \{j_1, j_2, \ldots\}$ is a finite set of jobs indexed in order of increasing arrival time (i.e., for $1 \leq i < |J| : A_i \leq A_{i+1}$). We assume that job parameters are revealed to a component only upon job arrival; i.e., a component does not have knowledge of future job arrivals. We call a job j_i *active* at time instant T, if $T \in [A_i, A_i + D_i)$. Let N be the maximum number of active jobs in the component at any given time.

The *monotonic absolute deadline* (MAD) [18] property of an aperiodic job states that if job j_i arrives before job j_k, then j_i's absolute deadline must occur before j_k's absolute deadline; more formally, $A_i \leq A_k \Leftrightarrow \bar{d}_i \leq \bar{d}_k$. Figure 3.1 provides a visual depiction of a legal MAD job arrival sequence. For *arbitrary aperiodic jobs*, the constraint of the MAD property is relaxed; that is, jobs may arrive in the system at any order of deadline (Figure 3.2).

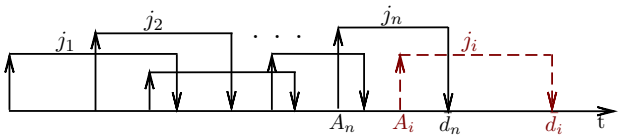


Figure 3.1:
MAD jobs: j_i has greater absolute deadline than j_n.

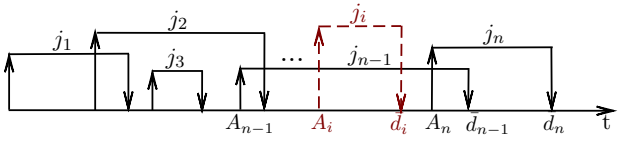


Figure 3.2:
Arbitrary jobs: j_i has smaller absolute deadline than j_{n-1}.

3.1.2 Sporadic Task Model

A **sporadic task** $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* e_i, a *(relative) deadline* d_i, and a *minimum inter-arrival separation* p_i, which is, for historical reasons, also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least p_i time units. Each job has a worst-case execution requirement equal to e_i and a deadline that occurs d_i time units after its arrival time. A *sporadic task system* $\tau \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_n\}$ is a collection of n such sporadic tasks. For each task $\tau_i \in \tau$, the task system is *constrained-deadline* if d_i less or equal p_i, *implicit-deadline* if d_i equals p_i, and *arbitrary-deadline* if d_i can be greater or equal p_i. A useful metric for a sporadic task τ_i is the *task utilization* $u_i \stackrel{\text{def}}{=} e_i/p_i$. The system utilization is denoted $U_\tau \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} u_i$.

Let p_{max} represent maximum period among all the task periods, and d_{max} represent maximum relative deadline among all task deadlines. For any task $\tau_i \in \tau$, we denote j-th job of τ_i by $\tau_{i,j}$, its arrival time as a_{ij} and absolute deadline as $\bar{d}_{ij} (= a_{ij} + p_i)$.

3.2 Workload Functions

The *workload* of a system represents the amount of work needs to be done by the processor in a given period of time. For determining schedulability of a real-time system, it is often useful to quantify the maximum amount of execution required by the tasks in the system that must complete over any time interval. A workload function depends on the scheduling algorithm used to allocate resource among the tasks. We denote an absolute time by T and a time interval-length by t.

3.2.1 Aperiodic Jobs

Given a set of aperiodic jobs J in the system, we can accurately quantify the maximum allowable workload over any interval of time.

Definition 1 (Demand) *For any J and time instant $T_1, T_2 \in \mathbb{R} : 0 \leq T_1 < T_2$, the function* $\mathsf{demand}(J, T_1, T_2)$ *represents the maximum cumulative execution requirement of all jobs in J that have both an arrival time and deadline in the interval $[T_1, T_2]$.*

$$\mathsf{demand}(J, T_1, T_2) = \sum_{\substack{j_i \in J: \\ (A_i \geq T_1) \wedge (A_i + D_i \leq T_2)}} E_i. \tag{3.1}$$

3.2.2 Sporadic Tasks

A sporadic task τ_i generates potentially infinite sequence of jobs. The workload for such task can be characterized for different scheduling schemes used by the task system τ as described below.

Earliest-Deadline First Scheduling

In earliest-deadline-first (EDF) scheduling algorithm, at any instant of time the scheduler chooses to execute the job of a task which has earliest deadline in time. To quantify the workload of an EDF-scheduled system, researchers [21] have derived the *demand-bound function*, defined below.

Definition 2 (Demand-Bound Function) *For any $t > 0$ and task τ_i, the* **demand-bound function** *(*dbf*) quantifies the maximum cumulative execution requirement of all jobs of τ_i that could have both an arrival time and deadline in any interval of length t. Baruah et al. [21] have shown that, for sporadic tasks,* dbf *can be calculated as follows.*

$$\mathsf{dbf}(\tau_i, t) = \max\left(0, \left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \cdot e_i. \tag{3.2}$$

Figure 3.3 gives a visual depiction of the demand-bound function for a sporadic task τ_i. Observe from the above definition and Figure 3.3 that the dbf is a right continuous function with discontinuities at time points of the form $t \equiv d_i + a \cdot p_i$ where $a \in \mathbb{N}$. The *cumulative demand-bound function* for task system τ is defined as follows:

$$\mathsf{DBF}(\tau, t) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \mathsf{dbf}(\tau_i, t) \tag{3.3}$$

It has been shown [21] that the condition $\mathsf{DBF}(\tau, t) \leq t, \forall t \geq 0$ is necessary and sufficient for sporadic task system τ to be EDF-schedulable upon a preemptive uniprocessor platform of unit speed. Furthermore, it has also been shown that the aforementioned condition needs to be verified at only time points in the following *ordered* set (elements are in non-decreasing order). These points correspond to the points of discontinuity for DBF.

$$\mathsf{TS}(\tau) \stackrel{\text{def}}{=} \bigcup_{\tau_i \in \tau} \{t \equiv d_i + a \cdot p_i \mid (a \in \mathbb{N}) \wedge (t \leq H_\tau)\}. \tag{3.4}$$

where H_τ is an upper bound on the maximum time instant that the schedulability condition must be verified at. For EDF-scheduled sporadic task systems on preemptive unit-speed processors, H_τ is at most $\mathsf{lcm}_{\tau_i \in \tau}\{p_i\}$. The above set is known as the **testing set** for sporadic task system τ. For any $t_a \in \mathsf{TS}(\tau)$, $t_a \leq t_{a+1}$ ($a \in \mathbb{N}^+$); if t_a is the last element of the set, we use the convention that t_{a+1} equals ∞. Also, we will assume that t_0 is equal to zero.

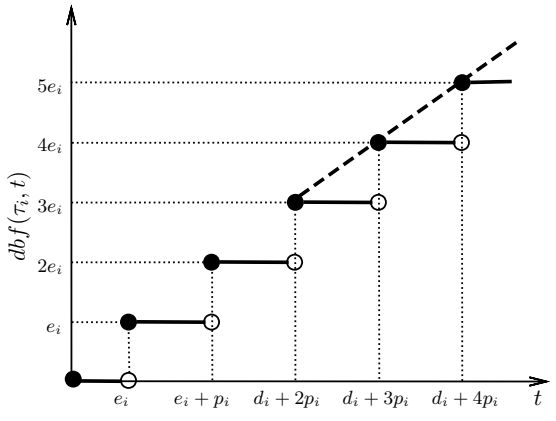


Figure 3.3: Demand-bound function $\mathsf{dbf}(\tau_i, t)$ of task τ_i

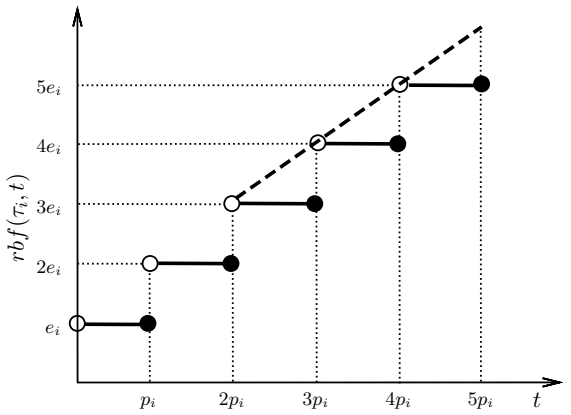


Figure 3.4: Request-bound function $\mathsf{rbf}(\tau_i, t)$ of task τ_i

Fixed-Priority Scheduling

When tasks are scheduled by a fixed-priority scheduling algorithm, we assume that task priorities are preassigned and each task has a fixed-priority. Tasks are indexed in non-increasing priority order. That is, τ_i has higher (or equal) priority than τ_j, if and only if, $i \leq j$. As tasks generate jobs, each job inherits the priority of its generating task. At any instant of time the scheduler chooses to execute the job of a task which has higher (preassigned) priority. The optimal fixed-priority scheduling algorithm for constrained-deadline sporadic tasks is *deadline monotonic* (DM) [61], which assigns each task a priority equal to the inverse of its relative deadline (i.e., tasks with shorter relative deadlines have priority greater than tasks with longer relative deadlines).

For FP-scheduled system, *request-bound function* is used to represent the system workload.

Definition 3 (Request-Bound Function) *For any $t > 0$ and sporadic task τ_i, the* **request-bound function** *(rbf) quantifies the maximum cumulative execution requests that could be generated by jobs of τ_i arriving within a contiguous time-interval of length t. It has been shown that for sporadic tasks,* rbf *can be calculated as follows [56].*

$$\mathsf{rbf}(\tau_i, t) \stackrel{def}{=} \left\lceil \frac{t}{p_i} \right\rceil \cdot e_i. \tag{3.5}$$

The *cumulative request-bound function* for task τ_i is defined as follows:

$$\mathsf{RBF}(\tau_i, t) \stackrel{\text{def}}{=} e_i + \sum_{j=1}^{i-1} \mathsf{rbf}(\tau_j, t). \tag{3.6}$$

Audsley *et al.* [14] have given a necessary and sufficient condition for sporadic task system τ to be fixed-priority-schedulable upon a preemptive uniprocessor platform of unit speed: $\exists t \in (0, d_i]$ such that $\mathsf{RBF}(\tau_i, t) \leq t, \forall i$. Furthermore, it has also been shown [11] that this condition needs to be verified at only time points in the following *ordered* set:

$$\mathsf{TS}_i(\tau) \stackrel{\text{def}}{=} \left\{ t = b \cdot p_a : a = 1, \ldots, i; b = 1, \ldots, \left\lfloor \frac{d_i}{p_a} \right\rfloor \right\} \cup \{d_i\}. \tag{3.7}$$

The above set is known as the *testing set* for sporadic task τ_i. The size of this set is $\sum_{j=1}^{i} \left\lfloor \frac{d_i}{p_j} \right\rfloor$ which is dependent on the task periods, and requires pseudo-polynomial time schedulability test.

For server-based models, we consider each component C of the compositional real-time system as a sporadic task system τ. For interface-based models, we assume that jobs can arrive aperiodically for a component.

3.3 Resource Model

A resource model characterizes the processing resource supply to a system. In compositional setting, the resource platform is shared among the components via the specifications of a resource model. In Section 1.1.1 we introduced these concepts. In this section we characterize the supply function and interface model used in this thesis.

3.3.1 Server-Based Interface

A *periodic resource* [81, 82] denoted by $\Gamma = (\Pi, \Theta)$, characterizes a partitioned resource that guarantees allocations of Θ time units every Π time units, where a resource period Π is a positive integer and a resource allocation time Θ is a real number in $(0, \Pi)$. This model is generalized by [39] as *Explicit-Deadline Periodic (EDP)* resource model. An EDP resource, denoted by $\Omega = (\Pi, \Theta, \Delta)$, guarantees that a component C (task system τ in our setting) executed upon resource Ω will receive at least Θ units of execution between successive time points in $\{t \equiv t_0 + \ell\Pi \mid \ell \in \mathbb{N}\}$ where t_0 is some initial service start-time t_0 for the periodic resource. Furthermore, the Θ units of service must occur Δ units after each successive time point in the aforementioned set. Obviously, $\Theta \leq \Delta$; for this thesis, we will make the simplifying assumption that $\Delta \leq \Pi$, as well. Furthermore, we will assume in this thesis that each component C is a sporadic task system[1] τ scheduled by EDF or FP upon Ω. (From now on, we use τ in the context of component C).

Definition 4 (Supply-Bound Function) *For any $t > 0$, the* **supply-bound function** *(sbf) quantifies the minimum execution supply that a component executed upon periodic resource may receive over any interval of length t.*

Easwaran et al. [39] have quantified the supply-bound function for an EDP resource in the following (see Figure 4.1 in Chapter 4 for a graphical depiction of the function) equation:

[1] Observe this is not a restriction on the number of hierarchical levels for our results. Subcomponents may also be represented by sporadic tasks, and our results will apply without change.

$$\mathsf{sbf}(\Omega, t) = \begin{cases} y_\Omega \Theta + \max\left(0, t - x_\Omega - y_\Omega \Pi\right), & \text{if } t \geq \Delta - \Theta \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

where $y_\Omega = \left\lfloor \frac{t-(\Delta-\Theta)}{\Pi} \right\rfloor$ *and* $x_\Omega = (\Pi + \Delta - 2\Theta)$.

Note that we can obtain supply-bound function for periodic resource Γ by replacing Δ by Π in the above equation (Equation 3.8).

Since sbf is discontinuous at certain points, researchers [39, 82] have defined a linear lower-bound to simplify the supply function:

Definition 5 (Lower Supply-Bound Function)

$$\mathsf{lsbf}(\Omega, t) = \frac{\Theta}{\Pi}\left(t - \Pi - \Delta + 2\Theta\right). \tag{3.9}$$

lsbf is a linear interpolation of the lower portions of the "steps" in the sbf function. It has been shown [82] that $\mathsf{lsbf}(\Omega, t) \leq \mathsf{sbf}(\Omega, t)$ for all $t > 0$. See Figure 4.1 for a visual depiction of lsbf.

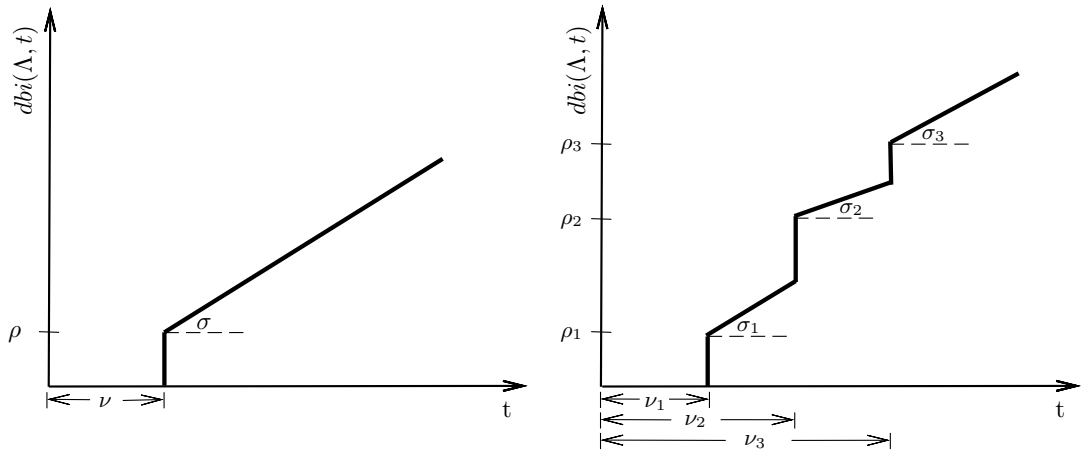
Similarly, a linear-upper bound of the sbf is given by the following definition (the concept will be useful in Chapter 5).

Definition 6 (Upper-Supply Bound Function)

$$\mathsf{usbf}(\Omega, t) \stackrel{def}{=} \frac{\Theta}{\Pi}(t - \Delta + \Theta). \tag{3.10}$$

usbf *is a linear interpolation of the upper portion of the "steps" of* sbf *(Fig. 4.1). From the definition it is evident that for all* $t > 0$, $\mathsf{usbf}(\Omega, t) \geq \mathsf{sbf}(\Omega, t)$.

Note that we use Θ to denote capacity in general, Θ^* to denote exact or minimal capacity required for a component to be schedulable, $\bar{\Theta}$ to denote sufficient capacity (Chapter 4), and $\widehat{\Theta}$ to denote approximate capacity obtained from our proposed algorithms EDFMINIMUMCAPACITY (Chapter 4) and FPMINIMUMCAPACITY (Chapter 5).



(a) Single-step demand interface (b) Arbitrary demand interface

Figure 3.5: Demand-curve interfaces

3.3.2 Demand-Curve Interface

Demand-curve interface for a component quantifies the precise demand of the workload for the component. There are standard techniques which uses real-time calculus to determine the demand-curve interfaces.

Definition 7 (Demand-Bound Curve) *A demand-bound curve of interface Λ gives an upper bound on the total demand of the set of jobs J admitted by a component. We denote the demand-bound curve for any interval of positive length t as* $\mathsf{dbi}(\Lambda, t)$. *Formally, the demand-bound curve* dbi *ensures that the following condition holds*

$$\forall T_1, T_2 \in \mathbb{R} : (0 \leq T_1 < T_2) :: \mathsf{demand}(J, T_1, T_2) \leq \mathsf{dbi}(\Lambda, T_2 - T_1). \tag{3.11}$$

We require that dbi *is a right continuous, piecewise linear, non-negative, and non-decreasing function of interval length $t \in \mathbb{R}_{\geq 0}$ ($t = T_2 - T_1$ in the above case).*

The above definition is applicable to existing real-time interface models such as RTC.

Single Step Demand Interface

A *single-step demand interface* (SSDI) can be defined as follows.

$$\mathsf{dbi}(\Lambda, t) \stackrel{\text{def}}{=} \begin{cases} 0, & t < \nu; \\ \rho + \sigma(t - \nu), & t \geq \nu. \end{cases} \tag{3.12}$$

Figure 3.5(a) gives a visual depiction of such interface. It is worthwhile to observe that the SSDI model is a generalization of the well-known bounded-delay resource model proposed by Mok et. al [69] (Section 2.1.1). In the above model, ν represents the "jitter" that an application executing upon the resource may experience, σ represents the "rate" of execution that the application is guaranteed over time and ρ represents initial supply of resource.

Arbitrary Demand Interface

An *arbitrary demand interface* can be considered as a sum of several single-step demand interfaces, with each step characterized by $(\sigma_i, \nu_i, \rho_i)$ as shown in Figure 3.5(b). To precisely model system demand, a cumulative demand-bound function (Equation 3.3) can be used as a demand interface for a system. Another example of demand interface is the periodic resource model described in Section 3.3.1.

CHAPTER 4: ALLOCATION: SERVER-BASED MODEL (EDF)

We address the MIB-RT problem in the context of server-based resource models (EDP) where components are sporadic tasks. In this chapter, we consider the case where component-level scheduling algorithm is earliest-deadline first (EDF). The contribution of this chapter can be summarized as follows:

> Given a sporadic task system τ as real-time component C running on an earliest-deadline periodic resource model Ω and accuracy parameter $\epsilon > 0$, we will obtain a real-time interface $I \in \mathcal{I}$ such that C is EDF-schedulable on I and $\beta(I^*) \leq \beta(I) \leq (1+\epsilon)\beta(I^*)$ where I^* is the interface with the (optimal) minimum interface bandwidth for C. Given Π, Δ, let Θ^* be the optimal minimum capacity for τ to be EDF-schedulable upon $\Omega^* = (\Pi, \Theta^*, \Delta)$. If our algorithm returns $\widehat{\Theta}$ for the given parameters, then $\Theta^* \leq \widehat{\Theta} \leq (1+\epsilon) \cdot \Theta^*$. Furthermore, our algorithm runs in time polynomial in the number of tasks in τ and $\frac{1}{\epsilon}$.

4.1 MIB-RT in Periodic Resources for EDF-Scheduled Components

In this section, we discuss previous work on MIB-RT for periodic resource model [63, 81].

4.1.1 An Exact Solution

For EDP resource Ω, Easwaran et al. [39] developed exact schedulability conditions when components are scheduled by EDF. The condition is given by the following theorem.

Theorem 1 (from [39]) *A sporadic task system τ is* EDF*-schedulable upon an* EDP *resource $\Omega = (\Pi, \Theta, \Delta)$, if and only if,*

$$(\mathsf{DBF}(\tau, t) \leq \mathsf{sbf}(\Omega, t),\ \forall t \leq H_\tau) \bigwedge \left(U_\tau \leq \frac{\Theta}{\Pi}\right) \tag{4.1}$$

where H_τ equals $\mathsf{lcm}_{\tau_i \in \tau}\{p_i\} + \max_{\tau_i \in \tau}\{d_i\}$.

This condition needs to be verified at each point in the testing set. As the size of the testing set is exponential (upto the lcm H_τ of task periods) in the number of tasks in the task system, the overall complexity of

the exact test is exponential.

4.1.2 An Over-provisioned Solution

Shin and Lee [83] proposed a sufficient approach for generating over-provisioned solutions given a component with an implicit-deadline sporadic task system τ (i.e., for all $\tau_i \in \tau$, $d_i = p_i$) and component-level scheduling algorithm EDF. We will paraphrase their result in the following theorem.

Theorem 2 (from [83]) *A component $C = (\tau, \mathsf{EDF})$ (where τ is an implicit-deadline sporadic task system) is EDF-schedulable upon periodic resource $\Gamma = (\Pi, \bar{\Theta})$ if there exists an $a \in \mathbb{N}^+$ such that $\bar{\Theta} \in [\theta_{\min}(a), \theta_{\max}(a)]$ where $\theta_{\min}(a) = \max\{\theta_0(a), \theta_1(a)\}$ and $\theta_{\max}(a) = \min\{\theta_2(a), \Pi\}$ with $\theta_0(a)$, $\theta_1(a)$, and $\theta_2(a)$ defined as follows.*

$$\theta_0(a) \stackrel{def}{=} \frac{(a+1)\Pi - p_{\min}(\tau)}{1 + \frac{a}{a+2}},$$

$$\theta_1(a) \stackrel{def}{=} \Pi \cdot \frac{(a+2)U_\tau}{a + 2U_\tau},$$

and

$$\theta_2(a) \stackrel{def}{=} \frac{(a+2)\Pi - p_{\min}(\tau)}{1 + \frac{a+1}{a+3}}.$$

Here $p_{\min}(\tau) = \min_{\tau_i \in \tau}\{p_i\}$. This approach has liner time complexity in determining capacity $\bar{\Theta}$, given a periodic resource with fixed period Π. In [47], we have shown that the capacity $\bar{\Theta}$ from this algorithm has an approximation ratio in the range $\frac{3}{2}$ to 3. Thus, the over-provisioned capacity obtained from this algorithm cannot guarantee that the returned $\bar{\Theta}$ satisfies $\Theta^* \leq \bar{\Theta} \leq (1+\epsilon)\Theta^*$ for any $\epsilon < \frac{1}{2}$. In the next section, we give our algorithm for determining minimum capacity $\widehat{\Theta}$ of an EDP resource within a user specified approximation ratio.

4.2 Approximate Solution

In this section we propose our approximate solution to MIB-RT problem in the context of EDF-scheduled components. Albers and Slomka [3] proposed the following approximation to dbf to reduce the number of discontinuities (and, thus, points in the testing set).

$$\widetilde{\mathsf{dbf}}(\tau_i, t, k) \stackrel{\text{def}}{=} \begin{cases} \mathsf{dbf}(\tau_i, t), & \text{if } t < d_i + (k-1)p_i; \\ u_i \cdot (t - d_i) + e_i, & \text{otherwise.} \end{cases} \tag{4.2}$$

The main intuition behind $\widetilde{\mathsf{dbf}}(\tau_i, t, k)$ is that it "tracks" dbf for exactly k discontinuities (i.e., "steps"). After k discontinuities, $\widetilde{\mathsf{dbf}}(\tau_i, t, k)$ is a linear interpolation of the subsequent discontinuous points (with slope equal to u_i). The steps with the thick lines and the sloped-dotted line in Figure 3.3 correspond to approximate demand $\mathsf{dbf}(\tau_i, t, 3)$ with $k = 3$. Let $\widetilde{\mathsf{DBF}}(\tau, t, k) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \widetilde{\mathsf{dbf}}(\tau_i, t, k)$. Albers and Slomka show [3], for any fixed $k \in \mathbb{N}^+$, the condition $\widetilde{\mathsf{DBF}}(\tau, t, k) \leq t, \forall t \geq 0$ is sufficient for sporadic task τ to be EDF-schedulable upon a preemptive uniprocessor platform of unit speed. The ordered testing set of this condition is reduced to

$$\widetilde{\mathsf{TS}}(\tau, k) \stackrel{\text{def}}{=} \bigcup_{\tau_i \in \tau} \left\{ t \equiv d_i + a \cdot p_i \mid (a \in \mathbb{N}) \wedge (a < k) \wedge (t \leq H_\tau) \right\}. \tag{4.3}$$

In order to obtain a fully polynomial-time approximation scheme for preemptive uniprocessors, Albers and Slomka [3] make the following observation regarding the relationship between dbf and $\widetilde{\mathsf{dbf}}$.

Lemma 1 (from [3]) *Given a fixed integer $k \in \mathbb{N}^+$, $\mathsf{dbf}(\tau_i, t) \leq \widetilde{\mathsf{dbf}}(\tau_i, t, k) \leq \left(\frac{k+1}{k}\right) \mathsf{dbf}(\tau_i, t)$ for all $\tau_i \in \tau$ and $t \in \mathbb{R}_{\geq 0}$.*

In addition to the above observation, we will now derive a technical lemma regarding $\widetilde{\mathsf{DBF}}$. Let $\langle (t, D_t), \alpha \rangle$ denote the half-line in Euclidean space $\mathbb{R}^2$, originating at point $(t, D_t) \in \mathbb{R}^2$ with slope α where $0 \leq \alpha \leq 1$ (i.e., $\langle (t, D_t), \alpha \rangle = \{(x, y) \in \mathbb{R}^2 \mid (x \geq t) \wedge (y = \alpha(x - t) + D_t))\}$). Additionally, we define the following function $\psi(\tau, t, k)$ which quantifies the slope of the expression $\widetilde{\mathsf{DBF}}(\tau, t, k)$ at any time t. Formally,

$$\psi(\tau, t, k) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau : t \geq d_i + (k-1)p_i} u_i. \tag{4.4}$$

Note that $\psi(\tau, t, \infty)$ is zero for all t. The following lemma states that for any element t_a of testing set $\widetilde{\mathsf{TS}}(\tau, k)$, the half-line defined by $\left\langle \left(t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)\right), \psi(\tau, t_a, k) \right\rangle$ lower bounds $\widetilde{\mathsf{DBF}}(\tau, t, k)$ for all t at least t_a.

Lemma 2 *Given any fixed $k \in \mathbb{N}^+ \cup \{\infty\}$, $t_a \in \widetilde{\mathsf{TS}}(\tau, k)$, and $t \geq t_a$*

$$\widetilde{\mathsf{DBF}}(\tau, t, k) \geq \psi(\tau, t_a, k) \cdot (t - t_a) + \widetilde{\mathsf{DBF}}(\tau, t_a, k). \tag{4.5}$$

Furthermore, for $t \in [t_a, t_{a+1})$, Equation 4.5 satisfies equality.

Proof: Given a k and t_a as defined above, consider $\widetilde{\mathsf{DBF}}(\tau, t, k)$ for any $t \geq t_a$.

$$\begin{aligned}
&\widetilde{\mathsf{DBF}}(\tau, t, k) \\
&= \textstyle\sum_{\tau_j \in \tau} \widetilde{\mathsf{dbf}}(\tau_j, t, k) \\
&= \textstyle\sum_{\tau_j \in \tau : t < d_j + (k-1)p_j} \widetilde{\mathsf{dbf}}(\tau_j, t, k) \\
&\qquad + \textstyle\sum_{\tau_j \in \tau : t \geq d_j + (k-1)p_j} [u_j(t - d_j) + e_j] \\
&\qquad\qquad \text{(by Definition of } \widetilde{\mathsf{dbf}}) \\
&\geq \textstyle\sum_{\tau_j \in \tau : t < d_j + (k-1)p_j} \widetilde{\mathsf{dbf}}(\tau_j, t_a, k) \\
&\qquad + \textstyle\sum_{\tau_j \in \tau : t \geq d_j + (k-1)p_j} [(u_j(t_a - d_j) + e_j) + u_j(t - t_a)] \\
&\qquad\qquad (\widetilde{\mathsf{dbf}} \text{ is monontonically non-decreasing}) \\
&= \textstyle\sum_{\tau_j \in \tau} \widetilde{\mathsf{dbf}}(\tau_j, t_a, k) + \psi(\tau, t_a, k) \cdot (t - t_a) \\
&\qquad\qquad \text{(by Definition of } \widetilde{\mathsf{dbf}} \text{ and } \psi).
\end{aligned}$$

The above series of inequalities show Equation 4.5. We now show that equality holds for Equation 4.5 for any time instant between t_a and t_{a+1}. For a task $\tau_j \in \tau$, consider $t, t' \in [d_j + (s-1)p_j, d_j + sp_j)$ where $s \in \mathbb{N}^+$ and $s \leq k - 1$. For any such t and t', $\widetilde{\mathsf{dbf}}(\tau_j, t, k)$ equals $\mathsf{dbf}(\tau_j, t)$ and $\widetilde{\mathsf{dbf}}(\tau_j, t', k)$ equals $\mathsf{dbf}(\tau_j, t')$, by definition of $\widetilde{\mathsf{dbf}}$ (Equation 4.2). Furthermore, by Definition 2, $\mathsf{dbf}(\tau_j, t)$ equals $\mathsf{dbf}(\tau_j, t')$ due to the floor in the expression of Equation 3.2. Thus, for such a t and t', $\widetilde{\mathsf{dbf}}(\tau_j, t, k)$ equals $\widetilde{\mathsf{dbf}}(\tau_j, t', k)$. In the third step of the derivation above the inequality may be replaced by equality for all $t \in [t_a, t_{a+1})$ and τ_j where $t < d_j + (k-1)p_j$, since there exists a $s \in \mathbb{N}^+$ $(s < k - 1)$ such that $t, t_a \in [d_j + (s-1)p_j, d_j + sp_j)$. (Otherwise, there would exist a $t_b (= d_\ell + (s-1)p_\ell) \in \widetilde{\mathsf{TS}}(\tau, k)$ for some $\tau_\ell \in \tau$ where $t_a < t_b < t_{a+1}$ which would contradict the ordering of the testing set). This implies that $\widetilde{\mathsf{dbf}}(\tau_j, t, k)$ equals $\widetilde{\mathsf{dbf}}(\tau_j, t_a, k)$ for all such $\tau_j \in \tau$ with $t < d_j + (k-1)p_j$. ■

The next corollary immediately follows by combining Lemmas 1 and 2.

Algorithm 1 Pseudo-code for determining minimum capacity for a periodic resource given Π, Δ, and τ. Note the algorithm is exact when k equals ∞.

EDFMINIMUMCAPACITY(Π, Δ, τ, k)

$\widehat{\Theta} \leftarrow U_\tau \cdot \Pi$
for each $t \in \widetilde{\mathsf{TS}}(\tau, k)$ $\triangleright$ (In order)
  $\triangleright$ For testing set with H_τ defined as in Theorem 1.
    $D_t \leftarrow \widetilde{\mathsf{DBF}}(\tau, t, k)$
    $\alpha \leftarrow \psi(\tau, t, k)$
    $\Theta_t^{\min} \leftarrow \infty$
    for $\ell \leftarrow \max\left\{1, \left\lfloor \frac{t-\Delta}{\Pi} \right\rfloor\right\}$ **to** $\left(\left\lceil \frac{t+\Delta}{\Pi} \right\rceil - 1\right)$
      $\Theta_\ell^{\min} \leftarrow \max \left\{ \begin{array}{r} \alpha\Pi, \\ \frac{D_t - t + \ell\Pi + \Delta}{\ell+1}, \\ \frac{D_t}{\ell}, \\ \frac{D_t + \alpha((\ell+1)\Pi + \Delta - t)}{\ell + 2\alpha} \end{array} \right\}$
      $\Theta_t^{\min} \leftarrow \min\{\Theta_t^{\min}, \Theta_\ell^{\min}\}$
    end (of inner loop)
    $\widehat{\Theta} \leftarrow \max\{\widehat{\Theta}, \Theta_t^{\min}\}$
end (of outer loop)
return $\widehat{\Theta}$

Corollary 1 *Given any fixed $k \in \mathbb{N}^+$, $t_a \in \widetilde{\mathsf{TS}}(\tau, k)$, and $t \geq t_a$,*

$$\mathsf{DBF}(\tau, t) \geq \left(\frac{k}{k+1}\right) \cdot \left[\psi(\tau, t_a, k) \cdot (t - t_a) + \widetilde{\mathsf{DBF}}(\tau, t_a, k)\right]. \tag{4.6}$$

4.2.1 Approximate Capacity Determination

In Algorithm 1, we give pseudocode for our algorithm, EDFMINIMUMCAPACITY, for determining the minimum capacity required to correctly schedule task system τ according to EDF upon an EDP resource with given Π and Δ parameters. One of the input parameters is $k \in \mathbb{N}_+$. If k is some fixed value, then EDFMINIMUMCAPACITY returns an approximate value for the minimum capacity; however, if the input value of k is equal to ∞, then the returned value will be the actual minimum capacity.

The intuition behind algorithm EDFMINIMUMCAPACITY is as follows: for each value t in the testing set $\widetilde{\mathsf{TS}}(\tau, k)$ find the minimum capacity $\Theta_t^{\min}$ required to guarantee that the half-line $\langle (t, \widetilde{\mathsf{DBF}}(\tau, t, k)),$-$\psi(\tau, t, k)\rangle$ is completely beneath $\mathsf{sbf}((\Pi, \Theta_t^{\min}, \Delta), t)$. By Lemma 2, this half-line is equal to $\widetilde{\mathsf{DBF}}(\tau, t, k)$

until the next testing set time-point. So, any capacity greater than this minimum capacity $\Theta_t^{\min}$ will ensure that $\widetilde{\mathsf{DBF}}$ falls below sbf up until the next point in the testing set. If we set $\widehat{\Theta}$ to be the maximum of all these $\Theta_t^{\min}$ (Line 17 of EDFMINIMUMCAPACITY) and $U_\tau \cdot \Pi$, then we ensure that each "step" of $\widetilde{\mathsf{DBF}}(\tau, t, k)$ is strictly less than $\mathsf{sbf}((\Pi, \widehat{\Theta}, \Delta), t)$ and $U_\tau \leq \frac{\widehat{\Theta}}{\Pi}$. Since $\widetilde{\mathsf{DBF}}(\tau, t, k) \geq \mathsf{DBF}(\tau, t)$ for all t, this implies that the schedulability condition of Theorem 1 is satisfied; thus, τ is EDF-schedulable upon EDP resource $\Omega = (\Pi, \widehat{\Theta}, \Delta)$. (Note that if the algorithm returns $\widehat{\Theta} > \Delta$, we cannot guarantee that τ is schedulable upon any EDP resource with parameters Π and Δ executing upon a unit-speed processor).

4.2.2 Algorithm Complexity

The complexity of EDFMINIMUMCAPACITY depends almost entirely upon the cardinality of $\widetilde{\mathsf{TS}}(\tau, k)$. To see this, observe that the inner loop (Lines 4 to 25) has a constant number of iterations due to the fact that $0 \leq \left(\left\lceil \frac{t+\Delta}{\Pi} \right\rceil - 1\right) - \left\lfloor \frac{t-\Delta}{\Pi} \right\rfloor \leq 2$ (since $0 \leq \Delta \leq \Pi$). The work inside the inner loop takes constant time as well. For the outer loop (Lines 1 to 28), the algorithm iterates (in non-decreasing order) through the testing set $\widetilde{\mathsf{TS}}(\tau, k)$. Using a "heap-of-heaps" described by Mok [68], the time complexity to obtain an element of the testing set is $O(\log n)$. Since for each element of the testing set, $t \equiv d_i + ap_i$ for some $\tau_i \in \tau$ and $a \in \mathbb{N}$, setting D_t and α (Lines 3 and 4) may be done in constant time on each iteration of the outer loop. (D_t is increased by e_i from prior iteration and α is only increased by u_i, if $a = (k-1)$). Therefore, the runtime complexity of EDFMINIMUMCAPACITY is $O(|\widetilde{\mathsf{TS}}(\tau, k)| \cdot \log n)$. If $k = \infty$, then $|\widetilde{\mathsf{TS}}(\tau, \infty)| \leq H_\tau$ which is potentially exponential in the number of tasks. The complexity for exactly determining the minimum capacity is, thus, the same complexity as the test of Theorem 1 on a fixed Ω. Otherwise, if k is a fixed integer, $|\widetilde{\mathsf{TS}}(\tau, k)| \leq kn$ and the complexity is $O(kn \log n)$.

4.2.3 Algorithm Correctness

To prove the correctness of EDFMINIMUMCAPACITY, we will show the following theorem which states that the value returned by the algorithm (i.e., $\widehat{\Theta}$) is at least the optimal minimum capacity value $\Theta^*(\Pi, \Delta, \tau)$. Furthermore, if the input k equals ∞, then the returned capacity is optimal.

Theorem 3 *For all $k \in \mathbb{N}_+ \cup \{\infty\}$,* EDFMINIMUMCAPACITY *returns $\widehat{\Theta} \geq \Theta^*(\Pi, \Delta, \tau)$. Furthermore, if $k = \infty$, $\widehat{\Theta} = \Theta^*(\Pi, \Delta, \tau)$.*

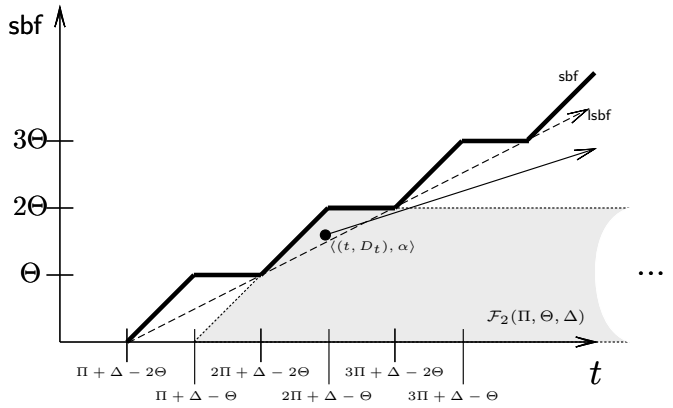


Figure 4.1: The solid line "step" function is sbf for Ω. The dashed line is the lower supply-bound function lsbf. The shaded region represents the ℓ-Feasibility Region for $\ell = 2$ with element $\langle (t, D_t), \alpha \rangle$.

In order to prove the above theorem, we require some additional definitions. The next definition quantifies the minimum capacity $\Theta(\leq \Delta)$ that is required for sbf to upper-bound a half-line $\langle (t, D_t), \alpha \rangle$. Definition 8 and 10 uses the function infimum $(\inf)$ instead of minimum $(\min)$. We will use the convention that $\inf$ returns ∞ on an empty set.

Definition 8 (Minimum Capacity for $\langle (t, D_t), \alpha \rangle$)

$$\Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) \stackrel{def}{=} \inf \left\{ \Theta \in \mathbb{R}_+ \;\middle|\; \begin{array}{ll} & (\Theta \leq \Delta) \\ \wedge & (\forall x \geq t : \alpha(x - t) + D_t \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)) \end{array} \right\}. \quad (4.7)$$

Since sbf is not a continuous function, it is difficult to calculate $\Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ in a straightforward linear way. Instead, we break the region below the sbf function into overlapping subregions we call *ℓ-feasibility regions* for any given $\Omega = (\Pi, \Theta, \Delta)$. An ℓ-feasibility region is the area below the ℓ^{th} "step" of the sbf function extending downward and infinitely to the right. Figure 4.1 gives a visual depiction; the definition below formalizes the ℓ-feasibility region concept.

Definition 9 (ℓ-Feasibility Region of Ω)

$$\mathcal{F}_\ell(\Pi,\Theta,\Delta) \stackrel{def}{=} \left\{ \langle (t,D_t),\alpha\rangle \in \mathbb{R}^2_{\geq 0}\times\mathbb{R}_{\geq 0} \;\middle|\; \begin{array}{ll} & \left(0 \leq \alpha \leq \frac{\Theta}{\Pi}\right) \\ \wedge & \left(\Theta \geq \frac{D_t - t + \ell\Pi + \Delta}{\ell+1}\right) \\ \wedge & \left(\Theta \geq \frac{D_t}{\ell}\right) \\ \wedge & \left(\Theta \geq \frac{D_t + \alpha((\ell+1)\Pi + \Delta - t)}{\ell + 2\alpha}\right) \end{array} \right\}. \tag{4.8}$$

The next defined function determines the minimum capacity for any given half-line $\langle (t,D_t),\alpha\rangle$ to be an element of the ℓ-feasibility region.

Definition 10 (ℓ-Minimum Capacity for $\langle (t,D_t),\alpha\rangle$)

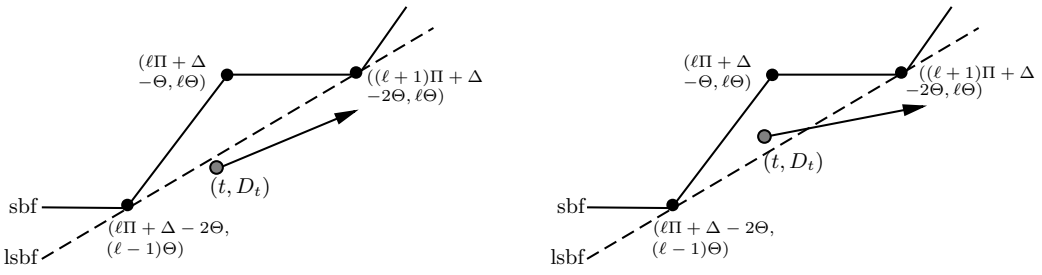
$$\Theta^*_\ell(\Pi,\Delta,\langle (t,D_t),\alpha\rangle) \stackrel{def}{=} \inf\left\{\Theta(\leq \Delta) \in \mathbb{R}_+ \mid \langle (t,D_t),\alpha\rangle \in \mathcal{F}_\ell(\Pi,\Theta,\Delta)\right\}. \tag{4.9}$$

We begin by showing that the half-line $\langle (t,D_t),\alpha\rangle$ is completely below sbf if and only if $\langle (t,D_t),\alpha\rangle$ is contained in some ℓ-feasibility region with $\ell > 0$.

Lemma 3 *For any $\langle (t,D_t),\alpha\rangle \in \mathbb{R}^2_{\geq 0}\times\mathbb{R}_{\geq 0}$, $\alpha(t'-t)+D_t \leq \mathsf{sbf}((\Pi,\Theta,\Delta),t')\ \forall t' \geq t$ if and only if there exists $\ell \in \mathbb{N}_+$ such that $\langle (t,D_t),\alpha\rangle \in \mathcal{F}_\ell(\Pi,\Delta,\Theta)$.*

Proof: We will give a geometric-based proof for this lemma. We will first show the "if" direction. Assume that $\langle (t,D_t),\alpha\rangle \in \mathcal{F}_\ell(\Pi,\Delta,\Theta)$ for some $\ell \in \mathbb{N}_+$. We must show that the half-line $\langle (t,D_t),\alpha\rangle$ is completely contained below the sbf function for $\Omega = (\Pi,\Theta,\Delta)$. (This is equivalent to showing $\alpha(t'-t)+D_t \leq \mathsf{sbf}((\Pi,\Theta,\Delta),t')\ \forall t' \geq t$).

If $D_t \leq \mathsf{lsbf}((\Pi,\Theta,\Delta),t)$, then the half-line $\langle (t,D_t),\alpha\rangle$ is below $\mathsf{lsbf}(\Omega,t')$ for all $t' \geq t$, because the slope of the half-line (i.e., α) is at most $\frac{\Theta}{\Pi}$ that is, the slope of $\mathsf{lsbf}(\Omega,t)$ (see Figure 4.2a). $\alpha \leq \frac{\Theta}{\Pi}$ follows from Equation 4.8. Since $\mathsf{lsbf}(\Omega,t') \leq \mathsf{sbf}(\Omega,t')$ for all $t' \geq t$, this implies that the half-line $\langle (t,D_t),\alpha\rangle$ never exceeds lsbf for $\Omega = (\Pi,\Theta,\Delta)$. Otherwise, if $D_t > \mathsf{lsbf}((\Pi,\Theta,\Delta),t)$, the originating point of the half-line $\langle (t,D_t),\alpha\rangle$ is contained within the triangular convex region defined by the (x,y)-points $(\ell\Pi+\Delta-2\Theta,(\ell-1)\Theta)$, $(\ell\Pi+\Delta-\Theta,\ell\Theta)$, and $((\ell+1)\Pi+\Delta-2\Theta,\ell\Theta)$; this region is formed by the intersection of the half-plane $y > \mathsf{lsbf}((\Pi,\Theta,\Delta),x)$ and the ℓ-feasibility region; e.g., see Figure 4.2b. From the figure it is obvious that this triangular region is contained below sbf. Furthermore, the entire



(a) $D_t \leq \text{lsbf}$ (b) $D_t > \text{lsbf}$

Figure 4.2: Illustration of Lemma 3. (a) Originating point of half-line $\langle (t, D_t), \alpha \rangle$ is below lsbf. (b) Originating point of $\langle (t, D_t), \alpha \rangle$ is above lsbf

ℓ-feasibility region is contained below sbf; thus, $D_t \leq \text{sbf}((\Pi, \Theta, \Delta), t)$. If we can show that the half-line $\langle (t, D_t), \alpha \rangle$'s value is below $\ell\Theta$ at $t_1 = (\ell+1)\Pi + \Delta - 2\Theta$, then $[t, t_1]$ interval portion of the half-line is contained entirely within the ℓ-feasibility region (and, thus, below sbf), and the remaining $[t_1, \infty)$ portion of the half-line falls below lsbf (and, thus, sbf). From the fourth condition of Equation 4.8, it must be that $\ell\Theta \geq D_t + \alpha((\ell+1)\Pi + \Delta - 2\Theta - t)$ which is equivalent to $\langle (t, D_t), \alpha \rangle$ being less than $\ell\Theta$ at t_1. Thus, in either case (based on the relative values of D_t and $\text{lsbf}(\Omega, t)$), we show the half-line $\langle (t, D_t), \alpha \rangle$ must be completely contained below the sbf function for Ω.

For the "only if" direction, we must show that if the half-line $\langle (t, D_t), \alpha \rangle$ is completely contained below the sbf function for Ω, then there exists an $\ell \in \mathbb{N}_+$ such that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_\ell(\Pi, \Theta, \Delta)$. Consider $\ell = \lceil \frac{D_t}{\Theta} \rceil$. (If $\Theta = 0$, then we will simply use $\ell = 0$, since in this case D_t must be zero due to $\text{sbf}((\Pi, 0, \Delta), t) = 0$ for all $t > 0$). The third condition of Equation 4.8 is trivially satisfied for this ℓ. It also must be true that $D_t > (\ell - 1)\Theta$. Thus, (t, D_t) must be below of the line defined by $y = x - (\ell\Pi + \Delta - (\ell+1)\Theta)$ (otherwise, (t, D_t) would be above the sbf function at t). This last constraint is equivalent to the second condition of Equation 4.8. The half-line's slope α obviously must not exceed $\frac{\Theta}{\Pi}$; otherwise, the $\langle (t, D_t), \alpha \rangle$ would eventually exceed sbf at some point. This constraint corresponds to the first condition of Equation 4.8. Finally, $\ell\Theta$ must be greater than the value of $\langle (t, D_t), \alpha \rangle$ at $t_1 = (\ell+1)\Pi + \Delta - 2\Theta$; otherwise, the half-line would exceed sbf at t_1. By the previous paragraph, we showed that this condition corresponds to satisfying the fourth condition of Equation 4.8. Therefore, for $\ell = \lceil \frac{D_t}{\Theta} \rceil$ we have satisfied all four conditions of Equation 4.8, implying that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\lceil \frac{D_t}{\Theta} \rceil}(\Pi, \Theta, \Delta)$. ■

Lemma 4 *For any $\ell \in \mathbb{N}_+$,*

$$\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) = \max \left\{ \begin{array}{r} \alpha\Pi, \\ \frac{D_t - t + \ell\Pi + \Delta}{\ell+1}, \\ \frac{D_t}{\ell}, \\ \frac{D_t + \alpha((\ell+1)\Pi + \Delta - t)}{\ell + 2\alpha} \end{array} \right\}. \tag{4.10}$$

Proof: Let Θ_{RHS} denote the right-hand side of Equation 4.10. We will show that both $\Theta_{\text{RHS}} \geq \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ and $\Theta_{\text{RHS}} \leq \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ which will imply the lemma. The inequality $\Theta_{\text{RHS}} \geq \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ follows from the observation that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_\ell(\Pi, \Theta_{\text{RHS}}, \Delta)$ (i.e., the four conditions of Equation 4.8 imply the lower bound).

We will show $\Theta_{\text{RHS}} \leq \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ by contradiction. Assume the negation of the inequality. In this case, since $\Theta_{\text{RHS}} > \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$, it must be that $\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ is strictly less than at least one of the four terms in the RHS of Equation 4.10; however, this implies that $\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_\ell(\Pi, \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle), \Delta)$. The previous statement contradicts the definition of Definition 10; thus, it must be that $\Theta_{\text{RHS}} \leq \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ is true. ■

Now that we know how to efficiently compute $\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ by simply checking the four values defined by the previous lemma, it would be convenient to use this value to compute the overall minimum capacity for half-line $\langle (t, D_t), \alpha \rangle$. The next lemma shows that this minimum bandwidth can be found by taking the minimum ℓ-minimum capacity for all positive ℓ.

Lemma 5

$$\Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) = \inf_{\ell > 0} \left\{ \Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) \right\}. \tag{4.11}$$

Proof: Let Θ_{RHS} denote the right-hand side of Equation 4.11. We will show that both $\Theta_{\text{RHS}} \geq \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ and $\Theta_{\text{RHS}} \leq \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ which will imply the lemma. First, we show $\Theta_{\text{RHS}} \geq \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$. By definition of infimum, for any $\delta > 0$, there exists $\ell \in \mathbb{N}_+$ such that $\{\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)\} \leq \Theta_{\text{RHS}} + \delta$. Definition 10 states that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_\ell(\Pi, \Theta_{\text{RHS}} + \delta, \Delta)$ for this ℓ. Lemma 3 then implies that $\alpha(t' - t) + D_t \leq \mathsf{sbf}((\Pi, \Theta_{\text{RHS}} + \delta, \Delta), t')$ for all $t' \geq t$. Therefore, for all $\delta > 0$, $\Theta_{\text{RHS}} + \delta$ must be in the set considered in the $\inf$ on the right-hand side of Equation 4.7 in Definition 8. Thus, $\Theta_{\text{RHS}} \geq \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$.

Next, we will show $\Theta_{\text{RHS}} \leq \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$. By Definition 8,

$$\forall x \geq t : \alpha \cdot (x - t) + D_t \leq \mathsf{sbf}((\Pi, \Delta, \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)), t).$$

Lemma 3 implies that there exists $\ell \in \mathbb{N}_+$ such that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_\ell(\Pi, \Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle), \Delta)$. By Definition 10, this implies that $\Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ is in the set considered in the right-hand side of Equation 4.9 which implies the inequality. ■

Fortunately, we do not need to evaluate $\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ for all positive ℓ to compute the minimum capacity for half-line $\langle (t, D_t), \alpha \rangle$. In the next lemma and corollary, we show that the smallest value of ℓ that needs to be considered is $\max\left(1, \left\lfloor \frac{t-\Delta}{\Pi} \right\rfloor\right)$ for a given Π and Δ. For all values of ℓ' smaller than this value, we can show that $\Theta^*_{\ell'}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) \geq \Theta^*_{\max\left(1, \left\lfloor \frac{t-\Delta}{\Pi} \right\rfloor\right)}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$.

Lemma 6 *For any $\ell, \ell' \in \mathbb{N}_+$ and $D_t, t, \alpha \in \mathbb{R}_{\geq 0}$, if $(t \geq \ell\Pi + \Delta)$, $(\ell' < \ell)$, and $(\alpha \leq 1)$, then*

$$[\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)] \Rightarrow [\langle (t, D_t), \alpha \rangle \in \mathcal{F}_\ell(\Pi, \Theta, \Delta)]. \tag{4.12}$$

Proof: Assume that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)$. We must show that all four conditions of Equation 4.8 are satisfied for $\mathcal{F}_\ell$. First note that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)$ implies that $\alpha \leq \frac{\Theta}{\Pi}$; thus, the first condition of Equation 9 is trivially satisfied for $\mathcal{F}_\ell$. By the third condition of Equation 4.8 for $\mathcal{F}_{\ell'}$, $D_t \leq \ell'\Theta$. Since $\ell > \ell'$, it must also be that $D_t \leq \ell\Theta$ (which satisfies third condition for $\mathcal{F}_\ell$). Similarly, the fourth condition of Equation 4.8 for $\mathcal{F}_{\ell'}$ implies that $\ell'\Theta \geq \alpha((\ell' + 1)\Pi + \Delta - 2\Theta - t) + D_t$. Since $\Theta \geq \alpha\Pi$ and $\ell \geq \ell'$, the fourth condition for $\mathcal{F}_\ell$ is also satisfied. Finally, consider the following expression

$$\begin{aligned}
&t - \ell\Pi - \Delta + (\ell + 1)\Theta \\
&\quad \geq (\ell\Pi + \Delta) - (\ell\Pi + \Delta) + (\ell + 1)\Theta \ \text{ (by assumption on } t) \\
&\quad = (\ell + 1)\Theta \\
&\quad > \ell'\Theta \\
&\quad \geq D_t \ \text{ (from condition three for } \mathcal{F}_{\ell'})
\end{aligned}$$

The above derivation implies the second condition for $\mathcal{F}_\ell$ which proves the lemma. ■

Given t, Π, and Δ, consider $\ell = \max\left(1, \lfloor \frac{t-\Delta}{\Pi} \rfloor\right)$. For any D_t, α, and $\ell' < \ell$ which satisfy the supposition of the above lemma, it must be that.

$$\begin{aligned}&\{\Theta(\leq \Delta) \in \mathbb{R}_+ \mid \langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)\} \\ &\quad \subseteq \left\{\Theta(\leq \Delta) \in \mathbb{R}_+ \mid \langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\max\left(1, \lfloor \frac{t-\Delta}{\Pi} \rfloor\right)}(\Pi, \Theta, \Delta)\right\}.\end{aligned}$$

By the above expression and Equation 4.9 of Definition 10, the corollary below follows immediately.

Corollary 2 *For any $\ell' \in \mathbb{N}_+$ and $D_t, t, \alpha \in \mathbb{R}_{\geq 0}$, if $\left(\ell' < \max\left(1, \lfloor \frac{t-\Delta}{\Pi} \rfloor\right)\right)$ and $(\alpha \leq 1)$, then*

$$\Theta^*_{\ell'}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) \geq \Theta^*_{\max\left(1, \lfloor \frac{t-\Delta}{\Pi} \rfloor\right)}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle).$$

In the following two lemmas and one corollary, we show that the largest value of ℓ that needs to be considered is $\lceil \frac{t+\Delta}{\Pi} \rceil - 1$ for a given Π and Δ. For all values of ℓ' larger than this value, we can show that $\Theta^*_{\ell'}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) \geq \Theta^*_{\lceil \frac{t+\Delta}{\Pi} \rceil - 1}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$.

Lemma 7 *For any $\ell, \ell' \in \mathbb{N}_+$ and $D_t, t, \alpha \in \mathbb{R}_{\geq 0}$, if $(t \leq \ell\Pi - \Delta)$, $(\ell' \geq \ell - 1)$, and $(\alpha \leq 1)$, then*

$$\left[\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)\right] \Rightarrow \left[\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'+1}(\Pi, \Theta, \Delta)\right].$$

Proof: Assume that $\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)$. This implies that at least one of the following is true:

$$\begin{aligned}&\alpha > \tfrac{\Theta}{\Pi} && (4.13a)\\ &\Theta < \tfrac{D_t - t + \ell'\Pi + \Delta}{\ell' + 1} && (4.13b)\\ &\Theta < \tfrac{D_t}{\ell'} && (4.13c)\\ &\Theta < \tfrac{D_t + \alpha((\ell'+1)\Pi + \Delta - t)}{\ell' + 2\alpha} && (4.13d)\end{aligned}$$

We will show that each of the above inequalities implies that $\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'+1}(\Pi, \Theta, \Delta)$. Obviously, Equation 4.13a trivially implies the aforementioned expression. If Equation 4.13b is true, then

$$\begin{aligned}&D_t > t - \ell'\Pi - \Delta + (\ell' + 1)\Theta \\ &\quad \Rightarrow D_t > t - (\ell' + 1)\Pi - \Delta + (\ell' + 2)\Theta \\ &\quad \Rightarrow \Theta > \tfrac{D_t - t + (\ell'+1)\Pi + \Delta}{\ell' + 2}\end{aligned}$$

The first implication above is due to the fact that right-hand side of the first inequality in the derivation is non-increasing in ℓ' (since $\Theta \leq \Pi$). Thus, if Equation 4.13b is true, then $\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'+1}(\Pi, \Theta, \Delta)$.

If Equation 4.13c is true, then consider the following expression:

$$\begin{aligned}
& t - (\ell'+1)\Pi - \Delta + (\ell'+2)\Theta \\
& \quad \leq \ell\Pi - \Delta - (\ell'+1)\Pi - \Delta + (\ell'+2)\Theta \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{(by supposition on } t\text{)} \\
& \quad \leq \ell\Pi - \ell\Pi - 2\Delta + (\ell'+2)\Theta \quad \text{(by } \ell' \geq \ell - 1\text{)} \\
& \quad \leq (\ell'+2)\Theta - 2\Theta \quad \text{(by } \Delta \geq \Theta\text{)} \\
& \quad = \ell'\Theta \\
& \quad < D_t \quad \text{(by Equation 4.13c)}
\end{aligned}$$

Thus, $\Theta < \frac{D_t - t + (\ell'+1)\Pi + \Delta}{\ell'+2}$ which implies $\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'+1}(\Pi, \Theta, \Delta)$.

Finally, if Equation 4.13d is true, then

$$\begin{aligned}
& \ell'\Theta < D_t + \alpha((\ell'+1)\Pi + \Delta - 2\Theta - t) \\
& \quad \Rightarrow \ell'\Theta < D_t + (\ell'+1)\Pi + \Delta - 2\Theta - t \\
& \quad \Rightarrow (\ell'+2)\Theta < D_t + (\ell'+1)\Pi + \Delta - t \\
& \quad \Rightarrow \Theta < \tfrac{D_t - t + (\ell'+1)\Pi + \Delta}{\ell'+2}
\end{aligned}$$

The first implication is due to $\alpha \leq 1$. The last inequality implies that when Equation 4.13d is true, $\langle (t, D_t), \alpha \rangle \notin \mathcal{F}_{\ell'+1}(\Pi, \Theta, \Delta)$; the lemma follows. ■

Lemma 8 *For any $\ell, \ell' \in \mathbb{N}_+$ and $D_t, t, \alpha \in \mathbb{R}_{\geq 0}$, if $(t \leq \ell\Pi - \Delta)$, $(\ell' > \ell - 1)$, and $(\alpha \leq 1)$, then*

$$[\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)] \Rightarrow [\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell-1}(\Pi, \Theta, \Delta)] \,.$$

Proof: By the contrapositive of Lemma 7, if $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)$, then $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'-1}(\Pi, \Theta, \Delta)$. If $\ell' - 1$ equals $\ell - 1$, then we have shown the lemma. Otherwise, the lemma follows from repeated application of the contrapositive of Lemma 7. ■

Given t, Π, and Δ, consider $\ell = \lceil \frac{t+\Delta}{\Pi} \rceil$. For any $D_t, \alpha, \ell' > \ell - 1$ which satisfy the supposition of the above lemma, it must be that.

$$\begin{aligned}&\{\Theta(\leq \Delta) \in \mathbb{R}_+ \mid \langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\ell'}(\Pi, \Theta, \Delta)\} \\ &\quad \subseteq \left\{\Theta(\leq \Delta) \in \mathbb{R}_+ \mid \langle (t, D_t), \alpha \rangle \in \mathcal{F}_{\lceil \frac{t+\Delta}{\Pi} \rceil - 1}(\Pi, \Theta, \Delta)\right\}.\end{aligned}$$

By the above expression and Equation 4.9 of Definition 10, the corollary below follows immediately.

Corollary 3 *For a given Π, Δ, and $\langle (t, D_t), \alpha \rangle$, the following is true for all $\ell' > \lceil \frac{t+\Delta}{\Pi} \rceil - 1$,*

$$\Theta^*_{\ell'}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) \geq \Theta^*_{\lceil \frac{t+\Delta}{\Pi} \rceil - 1}(\Pi, \Delta, \langle (t, D_t), \alpha \rangle).$$

Combining Corollaries 8 and 7 with Lemma 5, we obtain the following lemma which shows that $\Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ may be calculated with only a small number of values for ℓ.

Corollary 4

$$\Theta^*(\Pi, \Delta, \langle (t, D_t), \alpha \rangle) = \min_{\max\{1, (\lfloor \frac{t-\Delta}{\Pi} \rfloor)\} \leq \ell \leq (\lceil \frac{t+\Delta}{\Pi} \rceil - 1)} \{\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)\}.$$

The next lemma shows the maximum value of Θ obtained from calculating $\Theta^*(\cdot)$ for every value in the testing set $\widetilde{\mathsf{TS}}(\tau, k)$ may be used safely used as a capacity which will satisfy the condition of Theorem 1 (i.e., the EDP schedulability condition).

Lemma 9 *For any $t \geq 0$, $\widetilde{\mathsf{DBF}}(\tau, t, k) \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)$ and $U_\tau \leq \frac{\Theta}{\Pi}$, if and only if,*

$$\Theta \geq \max \left(\begin{array}{l} \max_{t \in \widetilde{\mathsf{TS}}(\tau,k)} \left\{ \Theta^* \left(\Pi, \Delta, \langle (t, \widetilde{\mathsf{DBF}}(\tau, t, k)), \psi(\tau, t, k) \rangle \right) \right\}, \\ U_\tau \cdot \Pi \end{array} \right). \tag{4.13}$$

Proof: We will show the "if" direction first, by showing its contrapositive. We must show that if either $U_\tau > \frac{\Theta}{\Pi}$ or $\exists t \geq 0 : \widetilde{\mathsf{DBF}}(\tau, t, k) > \mathsf{sbf}((\Pi, \Theta, \Delta), t)$, then the negation of the inequality of Equation 4.13 is also true. Assume that $U_\tau > \frac{\Theta}{\Pi}$. By the second expression in the outer "max" of Equation 4.13, the negation of the inequality is true. Now, assume there exists $t \geq 0$ such that $\widetilde{\mathsf{DBF}}(\tau, t, k) > \mathsf{sbf}((\Pi, \Theta, \Delta), t)$. There must exist two consecutive values t_a and t_{a+1} in $\widetilde{\mathsf{TS}}(\tau, k)$ where $t \in [t_a, t_{a+1})$. (Recall that if t_a is the largest value in $\widetilde{\mathsf{TS}}(\tau, k)$ then t_{a+1} is assumed to be ∞). Lemma 2 implies that for all $t' \in [t_a, t_{a+1})$,

$\widetilde{\mathsf{DBF}}(\tau, t', k)$ equals $\psi(\tau, t_a, k) \cdot (t' - t_a) + \widetilde{\mathsf{DBF}}(\tau, t_a, k)$. Since $t \in [t_a, t_{a+1})$,

$$\mathsf{sbf}((\Pi, \Theta, \Delta), t) < \psi(\tau, t_a, k) \cdot (t - t_a) + \widetilde{\mathsf{DBF}}(\tau, t_a, k).$$

By Lemma 3, this implies that for all $\ell \in \mathbb{N}_+$,

$$\langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k) \rangle \notin \mathcal{F}_\ell(\Pi, \Theta, \Delta).$$

By Equation 4.9 from Definition 10,

$$\Theta < \Theta^*_\ell(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k) \rangle)$$

for all $\ell \in \mathbb{N}_+$, which implies (by Lemma 5)

$$\Theta < \Theta^*(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k) \rangle).$$

Thus, $\Theta < \max_{t \in \widetilde{\mathsf{TS}}(\tau,k)} \left\{ \Theta^* \left(\Pi, \Delta, \langle (t, \widetilde{\mathsf{DBF}}(\tau, t, k)), \psi(\tau, t, k) \rangle \right) \right\}$. Again, the negation of the inequality of Equation 4.13 is true. The contrapositive of the "if" direction follows.

For the "only if" direction of the lemma, we will also consider the contrapositive. The contrapositive will follow by simply reversing the implications of the proof for the "if" direction. We give the argument for completeness. Assume that the negation of the inequality of Equation 4.13. If $\Theta < U_\tau \cdot \Pi$, then obviously $U_\tau > \frac{\Theta}{\Pi}$. Otherwise, if

$$\Theta < \max_{t \in \widetilde{\mathsf{TS}}(\tau,k)} \left\{ \Theta^* \left(\Pi, \Delta, \langle (t, \widetilde{\mathsf{DBF}}(\tau, t, k)), \psi(\tau, t, k) \rangle \right) \right\},$$

then there exists $t_a \in \widetilde{\mathsf{TS}}(\tau, k)$ such that $\Theta < \Theta^*(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k) \rangle)$. By Lemma 5,

$$\Theta < \Theta^*_\ell(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k) \rangle)$$

for all $\ell \in \mathbb{N}_+$. From Equation 4.9 from Definition 10, $\langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k) \rangle \notin \mathcal{F}_\ell(\Pi, \Theta, \Delta)$ for all $\ell \in \mathbb{N}_+$. By Lemma 3, there exists a $t \geq t_a$ such that $\mathsf{sbf}((\Pi, \Theta, \Delta), t) < \psi(\tau, t_a, k) \cdot (t - t_a) + \widetilde{\mathsf{DBF}}(\tau, t_a, k)$. Lemma 2 implies that for all $t' \geq t_a$, $\widetilde{\mathsf{DBF}}(\tau, t', k)$ is at least $\psi(\tau, t_a, k) \cdot (t' - t_a) +$

$\widetilde{\mathsf{DBF}}(\tau, t_a, k)$. Thus, since $t \geq t_a$, $\mathsf{sbf}((\Pi, \Theta, \Delta), t) < \widetilde{\mathsf{DBF}}(\tau, t, k)$. The contrapositive of the "only if" direction follows. ■

After proving the above lemma, we may now prove Theorem 3 which states that EDFMINIMUMCAPACITY returns a valid capacity (i.e., τ is schedulable upon the returned capacity) for finite k and an exact value for $k = \infty$.

Proof of Theorem 3 It is easy to see that $\widehat{\Theta}$ returned from $EDFMinimumCapacity$ corresponds to the value on the right-hand side of Equation 4.13 of Lemma 9; the loop from Line 1 to 28 iterate through each value t_a in $\widetilde{\mathsf{TS}}(\tau, k)$ (setting variables corresponding to $\widetilde{\mathsf{DBF}}(\tau, t_a, k)$ and $\psi(\tau, t_a, k)$ in Lines 3 and 4, respectively). The inner loop from Line 4 to 25 determines $\Theta^* \left(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k)\rangle\right)$; the implications of Corollary 4 and Lemma 5 show that we need to calculate $\Theta^*_\ell(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k)\rangle)$ only for values of ℓ from $\max\left(1, \lfloor \frac{t_a - \Delta}{\Pi} \rfloor\right)$ to $\lceil \frac{t_a + \Delta}{\Pi} \rceil - 1$. Finally, in Line 17, $\widehat{\Theta}$ is set to the maximum of $U_\tau \cdot \Pi$ and $\Theta^* \left(\Pi, \Delta, \langle (t_a, \widetilde{\mathsf{DBF}}(\tau, t_a, k)), \psi(\tau, t_a, k)\rangle\right)$ over all values t_a in $\widetilde{\mathsf{TS}}(\tau, k)$.

By Lemma 9, $\widetilde{\mathsf{DBF}}(\tau, t, k) \leq \mathsf{sbf}((\Pi, \widehat{\Theta}, \Delta), t)$ for all $t \geq 0$ and $U_\tau \leq \frac{\Theta}{\Pi}$. By Lemma 1, $\mathsf{DBF}(\tau, t) \leq \widetilde{\mathsf{DBF}}(\tau, t, k) \leq \mathsf{sbf}((\Pi, \widehat{\Theta}, \Delta), t)$ for all $t \geq 0$. Therefore, the supposition of Theorem 1 are satisfied and the τ will always meet all deadlines when scheduled by EDF upon $\Omega = (\Pi, \widehat{\Theta}, \Delta)$. When $k = \infty$, $\widetilde{\mathsf{DBF}}(\tau, t, k)$ equals $\mathsf{DBF}(\tau, t)$ for all $t \geq 0$; in this case, $\widehat{\Theta}$ equals $\Theta^*(\Pi, \Delta, \tau)$ (i.e., $\widehat{\Theta}$ is an exact value) due to the fact that both Lemma 9 and Theorem 1 are necessary and sufficient. ■

4.2.4 Approximation Ratio

In this section we prove that given an accuracy parameter ϵ, our algorithm returns minimum capacity with approximation ration $1 + \epsilon$. In the previous section, we have shown that EDFMINIMUMCAPACITY gives a valid answer when k is finite and an exact answer when k is infinite. When k is finite, we have not, yet, given any details on how accurate the returned $\widehat{\Theta}$ will be. In this section, we show that we may trade computational efficiency for accuracy; that is, as k increases the guaranteed accuracy of EDFMINIMUMCAPACITY increases along with its running time. Theorem 4 quantifies this tradeoff for a given k; Corollary 6 shows that this tradeoff permits an FPTAS for the $\mathsf{MIB\text{-}RT}$ problem. Before we prove Theorem 4 and Corollary 6, we need two technical lemmas.

Lemma 10 *Given Π, Δ, and τ, the following is true for all $k, \ell(\in \mathbb{N}_+)$, $t, D_t(\in \mathbb{R}_+)$, and $\alpha(\in [0,1])$,*

$$\Theta_\ell^*\left(\Pi, \Delta, \langle (t, D_t), \alpha\rangle\right) \leq \left(\tfrac{k+1}{k}\right) \cdot \Theta_\ell^*\left(\Pi, \Delta, \left\langle \left(t, \tfrac{k \cdot D_t}{k+1}\right), \tfrac{k\cdot\alpha}{k+1}\right\rangle\right). \tag{4.14}$$

Proof: By Lemma 4, $\Theta_\ell^*(\Pi, \Delta, \langle (t, D_t), \alpha\rangle)$ must be equal to one of the following: $\alpha\Pi$; $\frac{D_t - t + \ell\Pi + \Delta}{\ell+1}$; $\frac{D_t}{\ell}$; or $\frac{D_t + \alpha((\ell+1)\Pi + \Delta - t)}{\ell + 2\alpha}$. We will show that for each of the four possibilities, Equation 4.14 must hold. If $\Theta_\ell^*(\Pi, \Delta, \langle (t, D_t), \alpha\rangle)$ is equal to $\alpha\Pi$, then by Lemma 4,

$$\begin{aligned}
&\Theta_\ell^*\left(\Pi, \Delta, \left\langle \left(t, \frac{k \cdot D_t}{k+1}\right), \frac{k\cdot\alpha}{k+1}\right\rangle\right) \\
&\geq \left(\tfrac{k\cdot\alpha}{k+1}\right) \cdot \Pi \\
&= \left(\tfrac{k}{k+1}\right) \cdot \Theta_\ell^*\left(\Pi, \Delta, \langle (t, D_t), \alpha\rangle\right).
\end{aligned}$$

The last step implies Equation 4.14.

If $\Theta_\ell^*(\Pi, \Delta, \langle (t, D_t), \alpha\rangle)$ is equal to $\frac{D_t - t + \ell\Pi + \Delta}{\ell+1}$, we will consider two subcases: $t \leq \ell\Pi + \Delta$ and $t > \ell\Pi + \Delta$. If $t \leq \ell\Pi + \Delta$, then by Lemma 4,

$$\begin{aligned}
&\Theta_\ell^*\left(\Pi, \Delta, \left\langle \left(t, \frac{k \cdot D_t}{k+1}\right), \frac{k\cdot\alpha}{k+1}\right\rangle\right) \\
&\geq \tfrac{\frac{k\cdot D_t}{k+1} - t + \ell\Pi + \Delta}{\ell+1} \\
&\geq \left(\tfrac{k}{k+1}\right) \cdot \left(\tfrac{D_t - t + \ell\Pi + \Delta}{\ell+1}\right) \\
&= \left(\tfrac{k}{k+1}\right) \cdot \Theta_\ell^*\left(\Pi, \Delta, \langle (t, D_t), \alpha\rangle\right).
\end{aligned}$$

Otherwise, $t > \ell\Pi + \Delta$. In this case, Lemma 4 also implies

$$\begin{aligned}
&\Theta_\ell^*\left(\Pi, \Delta, \left\langle \left(t, \frac{k \cdot D_t}{k+1}\right), \frac{k\cdot\alpha}{k+1}\right\rangle\right) \\
&\geq \tfrac{\frac{k\cdot D_t}{k+1}}{\ell} \\
&\geq \tfrac{\left(\frac{k}{k+1}\right)\cdot D_t - \left(\frac{k}{k+1}\right)(t - \ell\Pi + \Delta)}{\ell+1} \\
&= \left(\tfrac{k}{k+1}\right) \cdot \left(\tfrac{D_t - t + \ell\Pi + \Delta}{\ell+1}\right) \\
&= \left(\tfrac{k}{k+1}\right) \cdot \Theta_\ell^*\left(\Pi, \Delta, \langle (t, D_t), \alpha\rangle\right).
\end{aligned}$$

Thus, in either subcase, Equation 4.14 holds.

If $\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ is equal to $\frac{D_t}{\ell}$, then Lemma 4 implies

$$\begin{aligned}
&\Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t, \frac{k \cdot D_t}{k+1} \right), \frac{k \cdot \alpha}{k+1} \right\rangle \right) \\
&\geq \frac{\frac{k \cdot D_t}{k+1}}{\ell} \\
&= \left(\frac{k}{k+1} \right) \cdot \Theta^*_\ell \left(\Pi, \Delta, \langle (t, D_t), \alpha \rangle \right).
\end{aligned}$$

Finally, if $\Theta^*_\ell(\Pi, \Delta, \langle (t, D_t), \alpha \rangle)$ is equal to $\frac{D_t + \alpha((\ell+1)\Pi + \Delta - t)}{\ell + 2\alpha}$, then Lemma 4 implies that

$$\begin{aligned}
&\Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t, \frac{k \cdot D_t}{k+1} \right), \frac{k \cdot \alpha}{k+1} \right\rangle \right) \\
&\geq \frac{\frac{k \cdot D_t}{k+1} - \left(\frac{k \cdot \alpha}{k+1} \right)((\ell+1)\Pi + \Delta - t)}{\ell + 2 \cdot \left(\frac{k \cdot \alpha}{k+1} \right)} \\
&\geq \left(\frac{k}{k+1} \right) \cdot \left(\frac{D_t - \alpha((\ell+1)\Pi + \Delta - t)}{\ell + 2\alpha} \right) \\
&= \left(\frac{k}{k+1} \right) \cdot \Theta^*_\ell \left(\Pi, \Delta, \langle (t, D_t), \alpha \rangle \right).
\end{aligned}$$

■

Lemma 11 *Given Π, Δ, and τ, the following is true for all $k \in \mathbb{N}_+$ and $t_a \in \widetilde{\mathsf{TS}}(\tau, k)$,*

$$\Theta^*(\Pi, \Delta, \tau) \geq \Theta^* \left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \widetilde{\mathsf{DBF}}(\tau, t_a, k)}{k+1} \right), \frac{k \cdot \psi(\tau, t_a, k)}{k+1} \right\rangle \right). \tag{4.15}$$

Proof: Let Θ_{RHS} denote the right-hand side of Equation 4.15. By definition of $\Theta^*(\Pi, \Delta, \tau)$ and Theorem 1,

$$\mathsf{sbf}((\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta), t) \geq \mathsf{DBF}(\tau, t) : \forall t \geq 0. \tag{4.16}$$

Now consider any $t_a \in \widetilde{\mathsf{TS}}(\tau, k)$. By combining Lemma 1 and Lemma 2, we have, for all $t \geq t_a$,

$$\left(\frac{k+1}{k} \right) \cdot \mathsf{DBF}(\tau, t) \geq \psi(\tau, t_a, k) \cdot (t - t_a) + \widetilde{\mathsf{DBF}}(\tau, t_a, k). \tag{4.17}$$

Combining the inequalities of Equations 4.16 and 4.17 gives us, for all $t \geq t_a$,

$$\mathsf{sbf}((\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta), t) \geq \left(\frac{k \cdot \psi(\tau, t_a, k)}{k+1} \right) \cdot (t - t_a) + \frac{k \cdot \widetilde{\mathsf{DBF}}(\tau, t_a, k)}{k+1}. \tag{4.18}$$

Lemma 3 and Equation 4.18 imply that there exists $\ell \in \mathbb{N}_+$ such that

$$\left\langle \left(t_a, \frac{k \cdot \widetilde{\mathsf{DBF}}(\tau, t_a, k)}{k+1} \right), \frac{k \cdot \psi(\tau, t_a, k)}{k+1} \right\rangle \in \mathcal{F}_\ell(\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta). \tag{4.19}$$

The above expression and Definition 10 imply

$$\Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \widetilde{\mathsf{DBF}}(\tau, t_a, k)}{k+1} \right), \frac{k \cdot \psi(\tau, t_a, k)}{k+1} \right\rangle \right) \leq \Theta^*(\Pi, \Delta, \tau). \tag{4.20}$$

The lemma follows from the expression above and Lemma 5. ■

The following corollary gives upper and lower bounds on the value obtained from the approximation. The corollary follows by combining Lemmas 10 and 11.

Corollary 5 *Given* Π*,* Δ*, and* τ*, the following is true for all* $k \in \mathbb{N}_+$ *and* $t \in \widetilde{\mathsf{TS}}(\tau, k)$*,*

$$\left(\tfrac{k+1}{k}\right) \cdot \Theta^*(\Pi, \Delta, \tau) \geq \min_{\ell \in \mathbb{N}_+} \left\{ \Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t, \widetilde{\mathsf{DBF}}(\tau, t, k) \right), \psi(\tau, t, k) \right\rangle \right) \right\}. \tag{4.21}$$

Finally, we give the theorem which quantifies the tradeoff between accuracy and computational complexity, in terms of k.

Theorem 4 *Given* Π*,* Δ*,* τ*, and* $k \in \mathbb{N}_+$*, the procedure* EDFMINIMUMCAPACITY *returns* $\widehat{\Theta}$ *such that*

$$\Theta^*(\Pi, \Delta, \tau) \leq \widehat{\Theta} \leq \left(\frac{k+1}{k} \right) \cdot \Theta^*(\Pi, \Delta, \tau).$$

Furthermore, EDFMINIMUMCAPACITY (Π, Δ, τ, k) *has time complexity* $O(kn \log n)$

Proof: Theorem 3 shows that $\Theta^*(\Pi, \Delta, \tau) \leq \widehat{\Theta}$; thus, we are left to prove the second inequality of the theorem. There are two cases that we will consider: $\widehat{\Theta} = U_\tau \cdot \Pi$ and $\widehat{\Theta} > U_\tau \cdot \Pi$. (Observe by Line 1, $\widehat{\Theta}$ must be at least $U_\tau \cdot \Pi$). In the case that $\widehat{\Theta}$ equals $U_\tau \cdot \Pi$, Theorem 1 implies that $\Theta^*(\Pi, \Delta, \tau)$ must be at least $U_\tau \cdot \Pi$. For this case, the second inequality follows, since $\frac{k+1}{k} \geq 1$ for all $k \in \mathbb{N}_+$.

In the case that $\widehat{\Theta}$ exceeds $U_\tau \cdot \Pi$, $\widehat{\Theta}$ must equal

$$\max_{t \in \widetilde{\mathsf{TS}}(\tau, k)} \left\{ \Theta^* \left(\Pi, \Delta, \langle (t, \widetilde{\mathsf{DBF}}(\tau, t, k)), \psi(\tau, t, k) \rangle \right) \right\}$$

according to the proof of Theorem 3. By Lemma 5, this is equivalent to

$$\max_{t \in \widetilde{\mathsf{TS}}(\tau,k)} \left\{ \min_{\ell \in \mathbb{N}_+} \left\{ \Theta_\ell^* \left(\Pi, \Delta, \langle (t, \widetilde{\mathsf{DBF}}(\tau,t,k)), \psi(\tau,t,k) \rangle \right) \right\} \right\}.$$

Applying Corollary 5 immediately yields the second inequality of the theorem for this case. ■

If we are given an accuracy parameter $\epsilon > 0$, we can appropriately set k equal to $\max\left(1, \lfloor \frac{1}{\epsilon} \rfloor\right)$ and guarantee an $(1+\epsilon)$ approximation ratio from the result returned from EDFMINIMUMCAPACITY. The following corollary (which follows from Theorem 4) shows that this approach yields an FPTAS for MIB-RT.

Corollary 6 *Given Π, Δ, τ, and $\epsilon > 0$, the procedure* EDFMINIMUMCAPACITY $\left(\Pi, \Delta, \tau, \lceil \frac{1}{\epsilon} \rceil\right)$ *returns* $\widehat{\Theta}$ *such that*

$$\Theta^*(\Pi, \Delta, \tau) \leq \widehat{\Theta} \leq (1+\epsilon) \cdot \Theta^*(\Pi, \Delta, \tau).$$

Furthermore, EDFMINIMUMCAPACITY $\left(\Pi, \Delta, \tau, \lceil \frac{1}{\epsilon} \rceil\right)$ *has time complexity* $O\left(\frac{n \log n}{\epsilon}\right)$.

4.3 Simulation Results

In this section, we show the simulation results for our proposed algorithm and compare it with the exact (Theorem 1) [39] and the sufficient algorithm (Theorem 2) [83]. During simulations, we have the following simulation parameters and value ranges:

1. The number of tasks (n) in the task system τ is taken from the set $\{2, 4, 6, 8, ...24\}$.

2. The system utilization U_τ is taken from the range $[0.1, 0.8]$ at 0.05-increments and individual task utilizations u_i are generated using UUniFast algorithm [25]. For a specific system utilization U_τ, this algorithm generates n random numbers in the range $[0, U_\tau]$ from uniform distribution in such a way that the sum of the numbers equals the system utilization U_τ.

3. Each sporadic task $\tau_i = (e_i, d_i, p_i)$ has a period p_i uniformly drawn from the interval $[5, 40]$. (A small period range is used to keep H_τ from becoming too large). The execution time e_i is set to $u_i \times p_i$. For each task, $d_i = p_i$.

4. The component level scheduling algorithm is EDF.

5. The value of k is taken from the set $\{1, 3, 5, ...19\}$; Π is taken from the set $\{5, 10, 15...40\}$; Δ is set equal to Π.

For each simulation, given task system size n and system utilization U_τ, we randomly generate task set parameters u_i, p_i, and e_i for each task τ_i. We implemented the linear time sufficient algorithm, the exact algorithm [39, 83] and EDFMINIMUMCAPACITY in MATLAB to generate sufficient, exact and approximate capacity, respectively. We have used two machines in our simulations with the following specifications: an Intel i-7 950 (four-core CPU with two hardware threads per core) machine with 6 GB RAM and a Phenom II X6 1100T (six-core CPU) machine with 8 GB RAM. To exploit the benefit of a multicore architecture, we have used 'parfor' function of MATLAB instead of a regular 'for' loop. Using this function, the simulation runs are concurrently executed on the cores of the CPU, which allowed us a significant speed-up in the duration of the simulations.

Our comparison metrics are relative error, testing set size and execution time of the algorithms. We compare relative error of our algorithm with the sufficient algorithm, and the other two metric of our algorithm with the exact algorithm. For our simulations, we vary the task system utilization (Figures 4.3, 4.4, and 4.10), resource period (Figures 4.7, 4.8, and 4.12), and task system size (Figures 4.5, 4.6, and 4.11) and evaluated each of the comparison metrics above. We also varied the approximation parameter k to evaluate the change in relative error (Figure 4.9). For the plots which vary the system utilization, resource period, or approximation parameter, each point represents the mean value of the comparison metric over 1000 simulation runs. For the plots which vary task system size, we reduced the number of simulation runs per point to 300. (Using 1000 runs for this experiment requires the simulation to execute for more than one week!). The 95% confidence intervals are shown for all the plots.

In Figure 4.3, relative error in calculation of capacity for the two algorithms are plotted as a function of task system utilization. In the graph, the dotted-line curve represents relative error for $\widehat{\Theta}$ and the solid-line curve represents relative error for $\bar{\Theta}$. For EDFMINIMUMCAPACITY, the mean relative error is less than 5%, whereas for the sufficient algorithm it ranges from 15% to 50%. Please note that approximation ratio for $\bar{\Theta}$ is between $3/2$ and 3 as shown in Section 4.1.2 whereas the approximation ratio for $\widehat{\Theta}$ is at most $4/3$ when $k = 3$ (from Theorem 4). Another interesting phenomena in the figure is that the relative error of sufficient algorithm reduces with increasing utilization. A potential explanation for the decrease in relative error is that some of the functions in the sufficient algorithm for setting the capacity (e.g., $\theta_0(a)$) do not

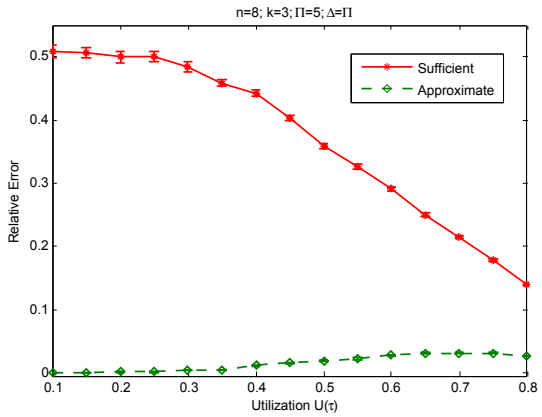


Figure 4.3: Relative Error vs System Utilization

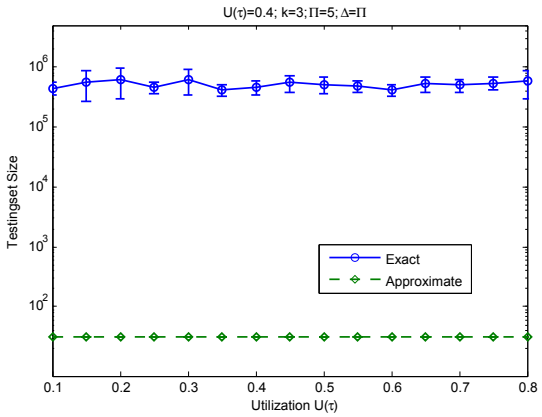


Figure 4.4: Testing Set Size vs System Utilization

depend on the utilization, only the task and resource period parameters. Such functions will be constant over increasing utilization while the optimal capacity must increase as utilization increases. The result is a reduction in the relative error of these (non-utilization-dependent) functions.

As we have mentioned before, the runtime complexity of EDFMINIMUMCAPACITY entirely depends on the size of the testing set. Figure 4.4 shows a logarithmic plot to compare between testing set sizes for exact algorithm ($|\mathsf{TS}(\tau)|$) and approximate algorithm ($|\widetilde{\mathsf{TS}}(\tau, k)|$). The solid-line curve in the graph represents $|\widetilde{\mathsf{TS}}(\tau, k)|$ and the dotted-line curve represents $|\mathsf{TS}(\tau)|$. As we know from the algorithm, $|\widetilde{\mathsf{TS}}(\tau, k)|$ depends on only the input k and task set size n (which is constant for our graph), on the other hand $|\mathsf{TS}(\tau)|$ may be exponentially large and has several orders of magnitude of variance, since it depends on the lcm of the periods. The sufficient algorithm is not shown since the algorithm only uses the utilization and minimum period parameters (i.e., it is linear time).

The next plot shows the effect of task system size on relative error for the two algorithms (Figure 4.5). In this plot we considered system utilization $U_\tau = 0.4$, approximation parameter $k = 3$, period $\Pi = 5$ and task system size is varied from 2 to 24. We observe that for EDFMINIMUMCAPACITY, task system size does not have significant effect on relative error, where as for the sufficient algorithm, relative error increases with task system size. This is due to the fact that in the later case, capacity depends on minimum period of the task system. As the task system size grows, minimum period tends to go smaller, which results the overestimation of sufficient capacity calculation. Figure 4.6 compares testing set size of the exact algorithm and the approximate algorithm for the same setting. Here, as expected, testing set size

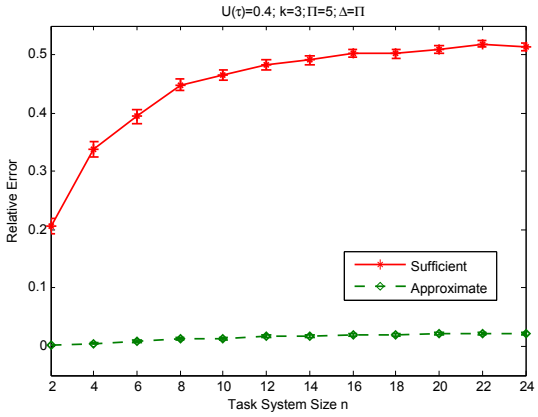


Figure 4.5: Relative Error vs Task System Size

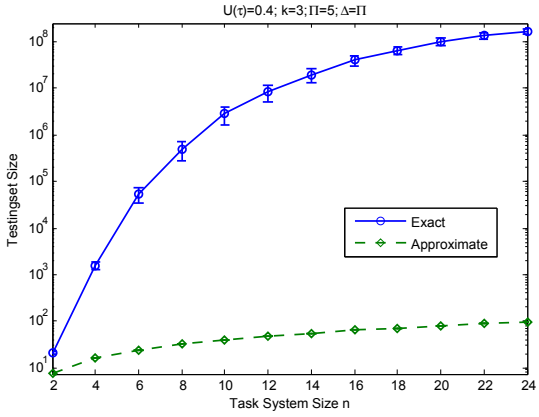


Figure 4.6: Testing Set Size vs Task System Size

of the exact algorithm grows exponentially with task system size compared to polynomial-time growth of EDFMINIMUMCAPACITY.

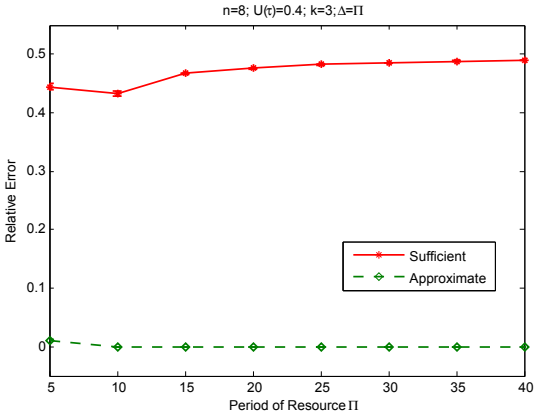


Figure 4.7: Relative Error vs Resource Period

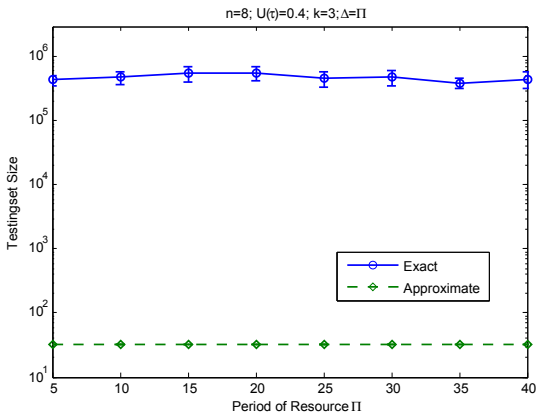


Figure 4.8: Testing Set Size vs Resource Period

The next two plots (Figures 4.7 and 4.8) compare the effect of resource period on the relative error and testing set size for the two algorithms. We varied Π from 5 to 40 and took $n = 8, U_\tau = 0.4$ and $k = 3$. We observe that varying Π does not effect relative error or testing set size for both the algorithms.

In Figure 4.9, the effect of approximation parameter (k) on relative error for the approximate algorithm (EDFMINIMUMCAPACITY) is shown. We observe that the relative error for $\widehat{\Theta}$ approaches 0 for $k \geq 7$. That is, the approximation gives as good as the exact performance for accuracy $\epsilon \geq \frac{1}{7}$.

Finally, in Figures 4.10, 4.11 and 4.12, we compare the execution time of our algorithm with the

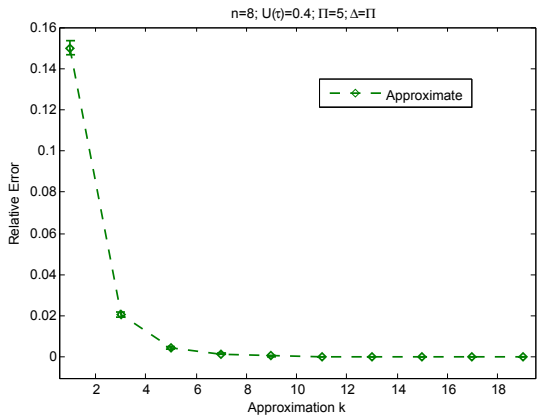


Figure 4.9: Relative Error vs Approximation (k)

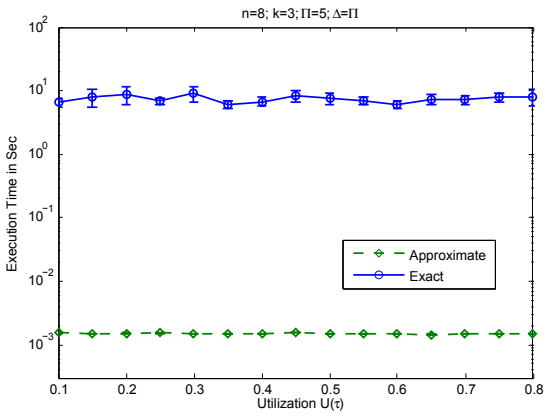


Figure 4.10: Execution Time vs System Utilization

exact algorithm. Observe that for each execution-time plot, the execution-time curves for each algorithm are roughly proportional to their corresponding testing set size plots. All three plots shows several orders of magnitude improvement in the approximate algorithm's execution time when compared with the exact algorithm's execution time.

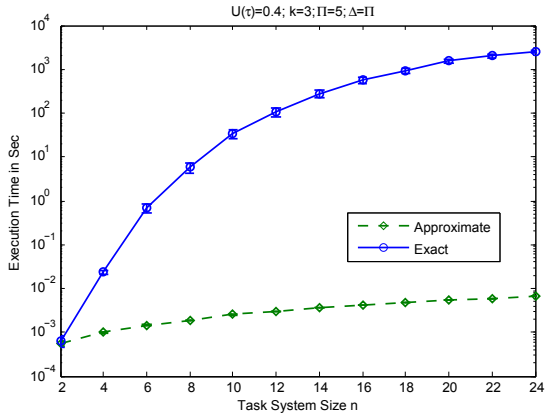


Figure 4.11: Execution Time vs Task System Size

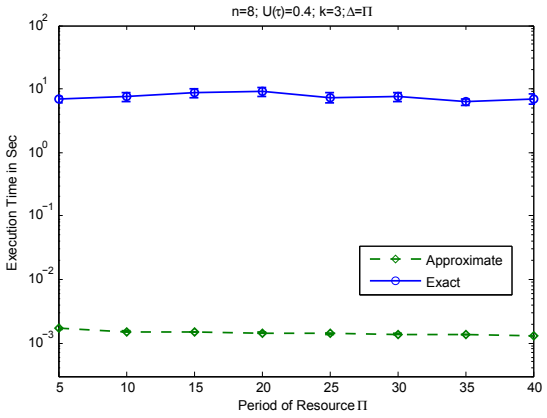


Figure 4.12: Execution Time vs Resource Period

Therefore, the simulation results strengthen the claim that EDFMINIMUMCAPACITY improves upon the performance of the sufficient algorithm while still maintaining a low polynomial runtime.

CHAPTER 5: ALLOCATION: SERVER-BASED MODEL (FP)

For EDP resource model with sporadic tasks as components, first we derive a response time based exact schedulability condition similar to dedicated uniprocessor scheduling by considering the "no-supply period" of the EDP resource as special higher priority task. We derive heuristics (i.e., lower bound and upper bound for response time) to efficiently perform the schedulability test similar to [32]. We develop a parametric approximation algorithm that addresses the current gap between computationally-expensive, exact solutions and computationally-inexpensive, sufficient solutions for MIB-RT problem. We claim the following.

> Given Π, Δ, task system τ, and accuracy parameter $\epsilon > 0$, let $\Theta^*(\Pi, \Delta, \tau)$ be the optimal minimum capacity for τ to be fixed-priority-schedulable upon EDP resource $\Omega^* = (\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta)$. Our algorithm returns $\widehat{\Theta}$ for the given parameters where $\Theta^*(\Pi, \Delta, \tau) \leq \widehat{\Theta} \leq (1 + \epsilon) \cdot \Theta^*(\Pi, \Delta, \tau)$. Furthermore, the time complexity of our algorithm is polynomial in the number of tasks in τ and $\frac{1}{\epsilon}$.

In other words, our algorithm is a fully-polynomial-time approximation scheme (FPTAS) for the MIB-RT problem with the *approximation ratio* $(1 + \epsilon)$. This implies that the system designer can pre-specify an arbitrary level of accuracy in obtaining solution to MIB-RT with the tunable algorithm. We also validate our algorithm by means of simulation over randomly generated task systems. One application of our approximation scheme is in thermally constrained real-time systems, where power-aware components dynamically tune the interface [49] to meet the temporal and thermal constrains.

In this chapter, we assume that each task in τ has a fixed-priority and task priorities are preassigned[1]. Tasks are indexed in non-increasing priority order; i.e., τ_i has higher (or equal) priority than τ_j, if and only if, $i \leq j$. As tasks generate jobs, each job inherits the priority of its generating task (i.e., all jobs generated by task τ_i have the same priority as τ_i). When multiple jobs of same task are ready to execute, the job with lowest index (arrival time) gets to execute first. Whenever component C is allocated the processor,

[1] See Audsley's paper [12] for an optimal priority assignment algorithm.

C executes the highest-priority job with remaining execution; ties are broken in favor of the job generated from the lower task index.

5.1 MIB-RT in Periodic Resources for FP-Scheduled Components

When components are scheduled by fixed-priority (dead-line monotonic), Easwaran et al. [39] developed exact schedulability conditions for EDP resource Ω. The condition is given by the following theorem

Theorem 5 (from [39]) *A sporadic task system τ is fixed-priority schedulable upon an* EDP *resource $\Omega = (\Pi, \Theta, \Delta)$, if and only if,*

$$(\forall i,\ \exists t \in (0, d_i] : \mathsf{RBF}(\tau_i, t) \leq \mathsf{sbf}(\Omega, t)) \bigwedge \left(U_\tau \leq \frac{\Theta}{\Pi} \right) \tag{5.1}$$

Since RBF is a non-decreasing function and changes only at the arrival of a job of τ_i or higher priority tasks, this conditions only needs to be verified at the testing set points defined by Equation 3.7 for each task in τ. The complexity of the exact test depends on the number of such points which can be pseudo polynomial in task parameters.

5.2 Determining Minimum Capacity Using Response Time

In this section, we derive an efficient exact schedulability test for a fixed-priority scheduled component upon periodic resources using similar approach as response time analysis of a dedicated uniprocessor system. Researchers defined response time for fixed-priority scheduled preemptive uniprocessor system consisting of periodic or sporadic tasks [11, 13, 51, 90]. The worst-case response time (WCET) R_i of task τ_i is computed using the following recursive function.

$$R_i = e_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil e_j \tag{5.2}$$

A fixed-priority scheduled component upon EDP resource Ω can be considered as an equivalent dedicated uniprocessor system by adding special higher priority tasks to the original task system which corresponds to the "no-supply period" of Ω. The traditional response time based fixed-priority schedulability

test can be applied to the modified task system.

Given a task system τ and EDP resource $\Omega \equiv (\Pi, \Theta, \Delta)$, we derive fixed-priority schedulability by considering two special periodic tasks τ_{-1} and τ_0 with higher priority then all the tasks in task system τ. Priority of τ_{-1} is higher than priority of τ_0. Let $\tau' = \{\tau \cup \tau_{-1} \cup \tau_0\}$ be the new task system with $\tau_{-1}(e_{-1}, d_{-1}, p_{-1}) \equiv (\Delta - \Theta, \Delta - \Theta, \infty)$ and $\tau_0(e_0, p_0, d_0) \equiv (\Pi - \Theta, \Pi, \Pi)$. Let τ_0 has a release jitter $j_0 = e_{-1}$, and for all the other task $\tau_i \in \tau'$, release jitter $j_i = 0$. This assumption enables us to correctly model the initial starvation period of $\Pi + \Delta - 2\Theta$ units of the periodic resource Ω. Clearly, these two tasks together represent the "no-supply period" of Ω, where τ_{-1} accounts for initial non-recurring starvation period $\Delta - \Theta$, and τ_0 accounts for the resource unavailability in every $t - (\Delta - \Theta)$ interval. The exact fixed-priority schedulability test of τ' is the reduction of the compositional schedulability test of τ with periodic resource Ω.

Now, to solve our problem of obtaining minimum capacity Θ, we use a binary search of Θ over the range $[0, \Pi]$, along with the exact fixed priority schedulability test of τ' shown in EXACTFPSCHEDULABILITY(τ, Ω).

The request bound function for the two special tasks τ_{-1} and τ_0 is as follows.

$$\mathsf{rbf}_0(t, \Omega) \stackrel{\text{def}}{=} (\Delta - \Theta) + \max\left\{0, \left\lceil \frac{t - (\Delta - \Theta)}{\Pi} \right\rceil \cdot (\Pi - \Theta)\right\}. \tag{5.3}$$

Using this and Equation 5.2, we determine response time for all tasks $\tau_i \in \tau$ as given by the following iterative equation.

$$R_i = e_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil e_j + \mathsf{rbf}_0(R_i, \Omega) \tag{5.4}$$

Thus, the exact schedulability test checks for each $\tau_i \in \tau'$, whether the response time R_i is less or equal its relative deadline d_i. In the next two subsections, we derive response time lower bound R_i^{lb} and upper bound R_i^{ub} for each task $\tau_i \in \tau'$, and use these heuristics to derive an efficient iterative schedulability test following the suggestions of [32]. Now, given fixed resource period Π and resource deadline Δ of Ω, we use a binary search of Θ over the range $[0, \Pi]$ to solve our problem of obtaining minimum capacity Θ along with the above exact test as shown in EXACTFPSCHEDULABILITY(τ, Ω).

Algorithm 2 Exact Fixed-Priority Schedulability algorithm considering τ_{-1}, τ_0 as higher priority tasks.

EXACTFPSCHEDULABILITY(τ, Ω)

1 $\tau_{-1}(e_{-1}, d_{-1}, p_{-1}) = (\Delta - \Theta, \Delta - \Theta, \infty)$; $\tau_0(e_0, d_0, p_0) = (\Pi - \Theta, \Delta, \Pi)$
2 $u_{-1} \leftarrow 0, u_0 \leftarrow e_0/p_0$
3 $R_0^{ub} \leftarrow e_{-1} + e_0$
4 **for** Each task $\tau_i \in \tau$ in descending order of priority
5 $\quad R_i^{ub} \leftarrow \frac{e_i + \sum_{\forall j \in hp(i)} e_j(1-u_j) + e_0(1-u_0) + (1-u_0)e_{-1}}{1 - \sum_{\forall j \in hp(i)} u_j - u_0}$
6 $\quad$ **if** $R_i^{ub} > d_i$
7 $\quad\quad R_i^{lb} \leftarrow \frac{e_i + \frac{\Theta}{\Pi}(\Delta - \Theta)}{\frac{\Theta}{\Pi} - U_{i-1}}$
$\quad\quad \triangleright$ Calculate initial value for response time of τ_i
8 $\quad\quad R_i \leftarrow \max\{(d_i + e_i)/2, d_i - R_{i-1}^{ub}, R_i^{lb}\}$
$\quad\quad \triangleright$ Solve the recurrence starting from initial response time
9 $\quad\quad R_i \leftarrow e_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil e_j + RBF_0(R_i, \Pi, \Theta, \Delta)$
10 $\quad\quad$ **if** $R_i > d_i$
11 $\quad\quad\quad$ **return** *Not Schedulable*.
12 $\quad$ **end if**
13 **end for**
14 **return** *Schedulable*.

5.2.1 Deriving Response Time Lower Bound

Observe from Equation 5.3, $\mathsf{rbf}_0(t, \Omega) = \Delta - \Theta$ when $t < \Delta - \Theta$. We do not have to consider this case, since the response time lower bound R_{-1}^{lb} for τ_{-1} must be equal to e_{-1}, which follows the response time lower bound for next highest priority job τ_0 to be at least $R_0^{lb} = e_{-1} + e_0$ at any time $t > 0$. All the jobs in τ have lower priority than these two jobs, thus the response time lower bound $R_i^{lb}, \forall \tau_i \in \tau$ must be greater R_0^{lb}, greater $\Delta - \Theta$.

For the case when $t \geq \Delta - \Theta$, we can derive response time lower bound R_i^{lb} by solving Equation 5.4.

Let $U_{i-1} = \sum_{\forall j \in hp(i)} \frac{e_j}{p_j}$ (excluding τ_{-1} and τ_0).

$$\begin{aligned} R_i = \; & e_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil e_j + (\Delta - \Theta) + \left\lceil \frac{R_i - (\Delta - \Theta)}{\Pi} \right\rceil (\Pi - \Theta) \\ & \geq e_i + \sum_{\forall j \in hp(i)} \frac{R_i}{p_j} e_j + (\Delta - \Theta) + \frac{R_i - (\Delta - \Theta)}{\Pi} (\Pi - \Theta) \\ & = e_i + R_i U_{i-1} + \frac{(\Delta - \Theta)\Pi + R_i(\Pi - \Theta) - (\Delta - \Theta)(\Pi - \Theta)}{\Pi} \\ & = \frac{\Pi e_i + R_i \Pi U_{i-1} + R_i(\Pi - \Theta) + \Theta(\Delta - \Theta)}{\Pi} \\ & = \frac{\Pi e_i + \Theta(\Delta - \Theta)}{\Theta - \Pi U_{i-1}} \\ & = \frac{e_i + \frac{\Theta}{\Pi}(\Delta - \Theta)}{\frac{\Theta}{\Pi} - U_{i-1}}. \end{aligned} \tag{5.5}$$

The last line of Equation 5.5 gives the lower bound R_i^{lb} for the response time of task τ_i.

5.2.2 Deriving Response Time Upper Bound

We derive the upper bound of response time for all $\tau_i \in \tau'$ using similar approach used by [23]. Using similar notation as [23], let $w_i(t)$ represents maximum amount of time that the processor executes τ_i in any interval length t, and $w_i^0(t)$ represents maximum amount of time that the processor executes τ_i in any interval length t when τ_i is the only task in the system. Then the worst case workload at time t for task τ_i is given by the following equation:

$$W_i(t) = \sum_{j=1}^{i} w_j(t) + w_0(t) + w_{-1}(t). \tag{5.6}$$

This implies:

$$W_i(t) \leq \sum_{j=1}^{i} w_j^0(t) + w_0^0(t) + w_{-1}^0(t). \tag{5.7}$$

Observe that for task τ_{-1}, $w_{-1}^0(t)$ is equal to $\min\{\Delta - \Theta, t\}$. Therefore, we obtain following bound.

$$w_{-1}^0(t) \leq \Delta - \Theta. \tag{5.8}$$

For τ_0, $w_0^0(t) = \min\left\{t - (\Delta - \Theta) - (p_0 - e_0)\left\lfloor \frac{t-(\Delta-\Theta)}{p_0} \right\rfloor, \left\lceil \frac{t-(\Delta-\Theta)}{p_0} \right\rceil e_0\right\}$, which can be upper bounded by linear approximation of the step function.

$$w_0^0(t) \leq u_0 t + e_0(1 - u_0) - (\Delta - \Theta)u_0. \tag{5.9}$$

For all other tasks in τ', w_i^0 can be upper bounded by the following linear approximation.

$$w_i^0(t) \leq u_i t + e_i(1 - u_i). \tag{5.10}$$

Combining Equation 5.8, 5.9, 5.10, we obtain workload upper bound for task τ_i at time t.

$$W_i(t) \leq \sum_{j=0}^{i}(u_j t + e_j(1 - u_j)) + (1 - u_0)(\Delta - \Theta). \tag{5.11}$$

Now, using the steps similar to Theorem 2 of [23], we obtain response time upper bound from workload upper bound.

$$R_i \leq \frac{e_i + \sum_{j=0}^{i-1} e_j(1 - u_j) + (1 - u_0)(\Delta - \Theta)}{1 - \sum_{j=0}^{i-1} u_j}. \tag{5.12}$$

Using Equation 5.5 and 5.12, we obtain initial value for our efficient iterative algorithm similar to [32]

$$R_i = \max\{(d_i + e_i)/2, d_i - R_{i-1}^{ub}, R_i^{lb}\} \tag{5.13}$$

In Algorithm 2, we give our efficient exact schedulability test for fixed-priority scheduled components upon periodic resources. We omit the proof of correctness for the algorithm as it directly follows from the derivations above.

5.3 Determining Minimum Capacity Using Testing Set

5.3.1 Exact Capacity Determination

The following theorem states the exact schedulability condition for EDP resource Ω, where task system is scheduled using fixed priority scheduling algorithm [39, 81, 82]. It says that for the task system τ to be schedulable with EDP resource, each task τ_i in τ must have a fixed point t before its deadline at which the cumulative request bound function for τ_i is less than the supply provided to the system at that point.

Theorem 6 (from [39]) *A sporadic task system τ is fixed priority schedulable upon an EDP resource $\Omega = (\Pi, \Theta, \Delta)$, if and only if,*

$$(\forall i,\ \exists t \in (0, d_i] : \mathsf{RBF}(\tau_i, t) \leq \mathsf{sbf}(\Omega, t)) \bigwedge \left(U_\tau \leq \frac{\Theta}{\Pi} \right) \tag{5.14}$$

In the next section we present an approximate algorithm to obtain minimum capacity for EDP resource when the component-level scheduling algorithm is fixed priority for the task system τ. We consider fixed period (Π) and deadline (Δ) for the EDP resource Ω.

5.3.2 Approximate Capacity Determination

We develop an approximate algorithm in the context of implicit or constrained deadline sporadic tasks scheduled by fixed-priority (e.g., DM, RM). Fisher and Baruah [45] proposed the following approximation to rbf (inspired by a similar approximation for EDF due to Albers and Slomka [3]) to reduce the number of points in the testing set.

$$\widetilde{\mathsf{rbf}}(\tau_i, t, k) \stackrel{\text{def}}{=} \begin{cases} \mathsf{rbf}(\tau_i, t), & \text{if } t \leq (k-1)p_i \\ e_i + \frac{t \cdot e_i}{p_i}, & \text{otherwise.} \end{cases} \tag{5.15}$$

This function tracks rbf for exactly $k-1$ steps and after the $k-1$-th step, it uses linear interpolation of subsequent discontinuous points of rbf (with slope equal to u_i). The steps in Figure 3.4 correspond to $\mathsf{rbf}(\tau_i, t, k)$, and the thick steps and the sloped-dashed line correspond to $\widetilde{\mathsf{rbf}}(\tau_i, t, k)$. The approximate cumulative request-bound function is defined as follows:

$$\widetilde{\mathsf{RBF}}(\tau_i, t) \stackrel{\text{def}}{=} e_i + \sum_{j=1}^{i-1} \widetilde{\mathsf{rbf}}(\tau_j, t, k). \tag{5.16}$$

For any fixed $k \in \mathbb{N}^+$, Fisher and Baruah [45] showed that if for all $\tau_i \in \tau$ there exists a $t \in (0, d_i]$ such that $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq t$ then the sporadic task system τ is static priority schedulable upon a preemptive uniprocessor platform of unit speed. The testing set for this condition is as follows:

$$\widehat{\mathsf{TS}}_i(\tau, k) \stackrel{\text{def}}{=} \{t = b \cdot p_a \mid a = 1, \ldots, i-1; b = 1, \ldots, k-1; t \in (0, d_i]\} \cup \{d_i\} \cup \{0\} \tag{5.17}$$

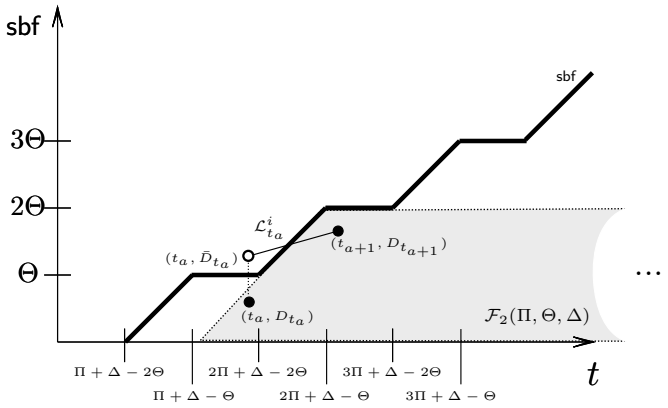


Figure 5.1: The solid line "step" function is sbf for Ω. The shaded region represents the ℓ-Feasibility Region $\mathcal{F}_2$ ($\ell = 2$). Line segment $\mathcal{L}^i_{t_a}$ intersects $\mathcal{F}_2$.

Let t_a, t_{a+1} denote any pair of consecutive values in the above ordered set.

Next, we give the relation between the request-bound function rbf and the approximate request-bound function $\widetilde{\mathsf{rbf}}$.

Lemma 12 (from [45]) *Given a fixed integer $k \in \mathbb{N}^+$, $\mathsf{rbf}(\tau_i, t) \leq \widetilde{\mathsf{rbf}}(\tau_i, t, k) \leq \left(\frac{k+1}{k}\right) \mathsf{rbf}(\tau_i, t)$ for all $\tau_i \in \tau$ and $t \in \mathbb{R}_{\geq 0}$.*

We will use this lemma in our approximation algorithm (Section 5.3.2).

Next, we define notation to represent the discontinuous line segments of the cumulative request-bound function ($\widetilde{\mathsf{RBF}}$). Let $\mathcal{L}^i_{t_a} \equiv \langle (t_a, \bar{D}_{t_a}), (t_{a+1}, D_{t_{a+1}}), \alpha \rangle$ be a line segment in the Euclidean space, $\mathbb{R}^2$, originating at open left end point $(t_a, \bar{D}_{t_a}) \in \mathbb{R}^2$ and ending at closed right end point $(t_{a+1}, D_{t_{a+1}}) \in \mathbb{R}^2$ with slope $\alpha \geq 0$ (Figure 5.1); more formally,

$$\mathcal{L}^i_{t_a} \stackrel{\text{def}}{=} \{(x, y) \in \mathbb{R}^2 \mid (x \in [t_a, t_{a+1}]) \wedge (y = \alpha(x - t_a) + \bar{D}_{t_a}))\}. \tag{5.18}$$

Please note the term α is included in the notation for convenience only; it is possible to determine the slope from points $(t_a, \bar{D}_{t_a})$ and $(t_{a+1}, D_{t_{a+1}})$ alone. We denote any point in the line segment by $(t, D_t) \in \mathcal{L}^i_{t_a}$.

$$\alpha \stackrel{\text{def}}{=} \sum_{\tau_h \in \tau : (t_a \geq (k-1)p_h) \wedge (h < i)} u_h. \tag{5.19}$$

The connection between $\mathcal{L}_{t_a}^i$ and $\widetilde{\mathsf{RBF}}$ is as follows. Consider a time $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$. Define D_{t_a} to be request-bound function at time t_a, that is $\widetilde{\mathsf{RBF}}(t_a)$ (Figure 3.4). At time t_a, some set of tasks with priority greater τ_i have job arrivals in the synchronous arrival sequence. Let $r_i(t)$ be the sum of the executions of these tasks. Formally,

$$r_i(t) \stackrel{\text{def}}{=} \sum_{\tau_j \in \tau : (j<i) \wedge (T_j \; divides \; t)} e_j. \tag{5.20}$$

At time t_a there is a discontinuity in the function $\widetilde{\mathsf{RBF}}$ in which $\widetilde{\mathsf{RBF}}$ increases by $r_i(t_a)$ and then is linear until the next discontinuity in $\widetilde{\mathsf{RBF}}$ (i.e., at time $t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$). Thus, $\widetilde{\mathsf{RBF}}$ is a line segment from t_a to t_{a+1} with slope equal to the total utilization of all task τ_j such that $j < i$ and $t_a \geq (k-1)T_j$. We denote $\bar{D}_{t_a}$ by the sum of request-bound function at time point t_a and job release at time t_a, that is, $\bar{D}_{t_a} = D_{t_a} + r_i(t_a)$. Finally, observe that

$$\alpha \stackrel{\text{def}}{=} \sum_{\tau_j \in \tau : (t_a \geq (k-1)T_j) \wedge (j<i)} u_j. \tag{5.21}$$

From the above definitions of t_a, t_{a+1}, D_{t_a}, $\bar{D}_{t_a}$, $D_{t_{a+1}}$, and α, it is straightforward to verify that the line segment $\mathcal{L}_{t_a}^i$ is equivalent to $(t, \widetilde{\mathsf{RBF}}(\tau_i, t))$ for all $t \in (t_a, t_{a+1}]$, with the exception at t_a where $\bar{D}_{t_a}$ is not equal to $\widetilde{\mathsf{RBF}}(t_a)$. From the definitions of t_a, t_{a+1}, D_{t_a}, $\bar{D}_{t_a}$, $D_{t_{a+1}}$, and α, the following lemma is apparent.

Lemma 13 *For any consecutive pairs of values* $(t_a, t_{a+1}) \in \widehat{\mathsf{TS}}_i(\tau, k)$, $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq D_t$ *for all* $(t, D_t) \in \mathcal{L}_{t_a}^i$.

We use the concept of ℓ-*feasibility region* of Ω similar to Definition 9 in Chapter 4 to define the region under the ℓ-th step of sbf. For our convenience, we redefine ℓ-feasibility region as follows:

Definition 11 (ℓ-Feasibility Region of Ω)

$$\mathcal{F}_\ell(\Pi, \Delta, \Theta) \stackrel{def}{=} \left\{ (t, D_t) \in \mathbb{R}^2_{\geq 0} \;\middle|\; \begin{array}{ll} & \left(\Theta \geq \frac{D_t - t + \ell\Pi + \Delta}{\ell + 1}\right) \\ \wedge & \left(\Theta \geq \frac{D_t}{\ell}\right) \end{array} \right\}. \tag{5.22}$$

Algorithm Description

In Algorithm 3, we present the pseudocode for our algorithm, FPMINIMUMCAPACITY[2]. Given task system τ and an EDP resource with Π and Δ as input, the algorithm returns approximate minimum capacity to correctly schedule the task system with the resource. The approximation parameter of the algorithm is the input $k \in \mathbb{N}^+$ ($k = \lceil \frac{1}{\epsilon} \rceil$). For some fixed k input, the algorithm returns the approximate minimum capacity; if k is equal to ∞, it returns exact minimum capacity. If FPMINIMUMCAPACITY returns a value $\Theta^{\min}$ that does not exceed Δ, then τ can be fixed-priority scheduled to meet all deadlines upon $\Omega = (\Pi, \Delta, \Theta^{\min})$. Note that the approximate capacity $\Theta^{\min}$ can be at most $(1 + \epsilon)$ times the exact capacity. If FPMINIMUMCAPACITY returns a capacity greater than Δ, then our algorithm cannot guarantee τ can be scheduled on any Ω with parameters Π and Δ. (Unless $k = \infty$, the algorithm is an approximation, and, thus, a returned capacity greater than Δ does not necessarily imply infeasibility of τ).

In our proposed algorithm, the objective is to compute minimum capacity $\Theta^{\min}$ for a task system τ such that τ is fixed-priority schedulable under EDP resource model. For each task $\tau_i \in \tau$, we find minimum capacity $\Theta_i^{\min}$ such that there exists a fixed point $t \in (0, d_i]$ at which the supply bound function sbf exceeds the cumulative request-bound function $\widetilde{\mathsf{RBF}}(\tau_i, t)$ (Theorem 6). To calculate $\Theta_i^{\min}$, we determine, for each consecutive pair of values (t_a, t_{a+1}) in the testing set $\widehat{\mathsf{TS}}_i(\tau, k)$, the minimum capacity $\Theta_{t_a}^{\min}$ required to guarantee that the line segment $\mathcal{L}_{t_a}^i$ is beneath $\mathsf{sbf}((\Pi, \Theta_{t_a}^{\min}, \Delta), t)$ for some $t \in (t_a, t_{a+1}]$. Since $\mathcal{L}_{t_a}^i$ is equivalent to $\widetilde{\mathsf{RBF}}$ for all $t \in (t_a, t_{a+1}]$, this implies that there exist a $t \in (t_a, t_{a+1}]$ such that $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta_{t_a}^{\min}, \Delta), t)$. To determine $\Theta_{t_a}^{\min}$, we take specific steps of the sbf (denote a selected step by ℓ) and determine the minimum Θ_ℓ such that some point of the line segment is below the ℓ-feasibility region with capacity Θ_ℓ. Each Θ_ℓ for (t_a, t_{a+1}) is set in lines 7, 8, 10 and 11. The following functions are used to determine the values of Θ_ℓ in our algorithm.

$$\begin{aligned} \Phi_1(\mathcal{L}_{t_a}^i, \ell, \Pi, \Delta) &\stackrel{\text{def}}{=} \tfrac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell + 1}, \\ \Phi_2(\mathcal{L}_{t_a}^i, \ell, \Pi, \Delta) &\stackrel{\text{def}}{=} \tfrac{\bar{D}_{t_a}}{\ell}, \\ \Phi_3(\mathcal{L}_{t_a}^i, \ell, \Pi, \Delta) &\stackrel{\text{def}}{=} \tfrac{\bar{D}_{t_a} + \alpha(\ell\Pi + \Delta - t_a)}{\ell + \alpha}. \end{aligned} \tag{5.23}$$

We will also show that we only need to consider the integer values of ℓ given by the following equations.

[2]We would like to thank the authors of [91] for identifying and correcting a small bug in our algorithm. In the original algorithm we did not check the condition in Line 9, and calculated $\Theta_{\lfloor \ell_1 \rfloor}$ and $\Theta_{\lceil \ell_2 \rceil}$ for all cases including the case $\lceil \ell_2 \rceil > \lfloor \ell_1 \rfloor$, which corresponds to an undefined interval (See Lemma 20, 21 and 22 for more details).

Algorithm 3 Pseudo-code for determining minimum capacity for a periodic resource given Π, Δ, and τ using fixed-priority scheduling algorithm. Note the algorithm is exact when k equals ∞.

FPMINIMUMCAPACITY(Π, Δ, τ, k)

$\Theta^{\min} \leftarrow U_\tau \cdot \Pi$
for each $\tau_i \in \tau$
  $\Theta_i^{\min} \leftarrow \infty$
  for each $(t_a, t_{a+1}) \in \widehat{\mathsf{TS}}_i(\tau, k)$ ▷ (In order)
    $\bar{D}_{t_a} \leftarrow \hat{W}_i(t_a) + r_i(t_a)$
    $D_{t_{a+1}} \leftarrow \hat{W}_i(t_{a+1})$
    $\Theta_{\lfloor \ell_1 \rfloor + 1} \leftarrow \Phi_1(\mathcal{L}_{t_a}^i, \lfloor \ell_1 \rfloor + 1, \Pi, \Delta)$
    $\Theta_{\lceil \ell_2 \rceil - 1} \leftarrow \Phi_2(\mathcal{L}_{t_a}^i, \lceil \ell_2 \rceil - 1, \Pi, \Delta)$
    if $\lceil \ell_2 \rceil \leq \lfloor \ell_1 \rfloor$
      $\Theta_{\lfloor \ell_1 \rfloor} \leftarrow \Phi_3(\mathcal{L}_{t_a}^i, \lfloor \ell_1 \rfloor, \Pi, \Delta)$
      $\Theta_{\lceil \ell_2 \rceil} \leftarrow \Phi_3(\mathcal{L}_{t_a}^i, \lceil \ell_2 \rceil, \Pi, \Delta)$
    else
      $\Theta_{\lfloor \ell_1 \rfloor}, \Theta_{\lceil \ell_2 \rceil} \leftarrow \infty$
    $\Theta_{t_a}^{\min} \leftarrow \min\{\Theta_{\lfloor \ell_1 \rfloor + 1}, \Theta_{\lceil \ell_2 \rceil - 1}, \Theta_{\lfloor \ell_1 \rfloor}, \Theta_{\lceil \ell_2 \rceil}\}$
    $\Theta_i^{\min} \leftarrow \min\{\Theta_i^{\min}, \Theta_{t_a}^{\min}\}$
  end (of inner loop)
  $\Theta^{\min} \leftarrow \max\{\Theta^{\min}, \Theta_i^{\min}\}$
end (of outer loop)
return $\Theta^{\min}$

$$\ell_1 \stackrel{\text{def}}{=} \frac{(t_{a+1} - \Delta) + \sqrt{(t_{a+1} - \Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi}, \tag{5.24}$$

$$\ell_2 \stackrel{\text{def}}{=} \frac{(t_a - \Delta) + \sqrt{(t_a - \Delta)^2 + 4\Pi \bar{D}_{t_a}}}{2\Pi}. \tag{5.25}$$

That is, we consider $\lfloor \ell_1 \rfloor, \lfloor \ell_1 \rfloor + 1, \lceil \ell_2 \rceil - 1$ and $\lceil \ell_2 \rceil$ to evaluate Θ_ℓ. The logic behind the choice of Φ functions and our definition of ℓ_1 and ℓ_2 will be more apparent in the proof of correctness section below.

Since we are looking for only one point in $t \in (0, d_i]$ for task τ_i where $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}(\Omega, t)$, we only need a single line segment of $\widetilde{\mathsf{RBF}}(\tau_i, t)$ that intersects with $\mathsf{sbf}(\Omega, t)$ and gives minimum capacity. Thus, we set $\Theta_i^{\min}$ to be the minimum of all $\Theta_{t_a}^{\min}$ values for each of the line segment of $\widetilde{\mathsf{RBF}}$. Finally, we set $\Theta^{\min}$ to be the maximum of all $\Theta_i^{\min}$ values. This ensures that for each task $\tau_i \in \tau$, we find a $t \leq d_i$ such that $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta^{\min}, \Delta), t)$. Since $\widetilde{\mathsf{RBF}}(\tau_i, t) \geq \mathsf{RBF}(\tau_i, t)$ for all t, this implies

Theorem 6; thus τ is fixed-priority schedulable upon EDP resource $\Omega = (\Pi, \Theta^{\min}, \Delta)$.

Algorithm Complexity

The complexity of FPMINIMUMCAPACITY depends on the number of tasks n in the task set τ and the cardinality of testing set $\widehat{\mathsf{TS}}_i(\tau, k)$ for each task τ_i. The outer loop of the algorithm (Lines 1 to 28) iterates for each task, thus n times in total. The inner loop (Lines 4 to 25) scans every pair of testing set points in $\widehat{\mathsf{TS}}_i(\tau, k)$ (in non-decreasing order) for task τ_i, and this can take at most $1+(i-1)(k-1)$ times for a single task. Using a "heap-of-heaps" described by Mok [68], the time complexity to obtain an element of the testing set is $O(\log n)$. Setting $\bar{D}_{t_a}$, $D_{t_{a+1}}$ and α (Lines 5, 6 and 4) is done in constant time on each iteration of the inner loop. Again, setting ℓ values and evaluating Θ values using these (Line 7 to 11) takes constant time. Therefore, the runtime complexity of FPMINIMUMCAPACITY is $O(\log n \cdot \sum_{i=1}^{n} |\widehat{\mathsf{TS}}_i(\tau, k)|)$. If $k = \infty$, the complexity for exactly determining the minimum capacity is the same complexity as the test of Theorem 6 on a fixed Ω, which may be pseudo-polynomial depending on the period of tasks. Otherwise, if k is a fixed integer, the complexity is at most $O\left(\log n \cdot \sum_{i=1}^{n}(1 + (i-1)(k-1))\right)$ times, which is $O(kn^2 \log n)$.

Algorithm Correctness

To prove the correctness of FPMINIMUMCAPACITY, we prove the following theorem which states that the value returned by the algorithm (i.e., $\Theta^{\min}$) is at least the optimal minimum capacity value $\Theta^*(\Pi, \Delta, \tau)$. Furthermore, if the input k equals ∞, then the returned capacity is optimal.

Theorem 7 *For all $k \in \mathbb{N}^+ \cup \{\infty\}$,* FPMINIMUMCAPACITY *returns $\Theta^{\min} \geq \Theta^*(\Pi, \Delta, \tau)$. Furthermore, if $k = \infty$, $\Theta^{\min} = \Theta^*(\Pi, \Delta, \tau)$.*

We require some additional definitions similar to [46] for notational convenience. The next definition quantifies the minimum capacity $\Theta(\leq \Delta)$ that is required for sbf to exceed the line segment $\mathcal{L}_{t_a}^i$ at some point (t, D_t). We will use the convention that $\inf$ returns ∞ on an empty set.

Definition 12 (Minimum Capacity for $\mathcal{L}_{t_a}^i$)

$$\Theta^*\left(\Pi, \Delta, \mathcal{L}_{t_a}^i\right) \stackrel{def}{=} \inf\left\{\Theta \in \mathbb{R}^+ \;\middle|\; \begin{array}{ll} & (\Theta \leq \Delta) \\ \wedge & \left(\exists (t, D_t) \in \mathcal{L}_{t_a}^i : D_t \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)\right) \end{array}\right\}. \tag{5.26}$$

The next function determines the minimum capacity for any given line segment $\mathcal{L}^i_{t_a}$ to have a point in the ℓ-feasibility region.

Definition 13 (ℓ-Minimum Capacity for $\mathcal{L}^i_{t_a}$)

$$\Theta^*_\ell\left(\Pi, \Delta, \mathcal{L}^i_{t_a}\right) \stackrel{def}{=} \inf\left\{ \Theta(\leq \Delta) \in \mathbb{R}^+ \;\middle|\; \begin{array}{l} \exists (t, D_t) \in \mathcal{L}^i_{t_a} \\ \quad : (t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta, \Delta) \end{array} \right\}. \tag{5.27}$$

Note the two above definitions use infimum, since they are defined over infinite sets; however, we will later see (Corollary 10) that the infimum corresponds to the minimum (i.e., the value returned by $\inf$ exists in the set specified in the right-hand side of Equations 5.26 and 5.27).

In order to prove Theorem 7, we must prove some additional lemmas. We start by presenting the three conditions on the value of Θ that are necessary and sufficient condition for a line segment $\mathcal{L}^i_{t_a}$ to have a point in the ℓ-feasibility region.

Lemma 14 *For any two consecutive pair of values $(t_a, t_{a+1}) \in \widehat{\mathsf{TS}}_i(\tau, k)$, there exists $(t, D_t) \in \mathcal{L}^i_{t_a}$ such that $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Delta, \Theta)$ for some $\ell \in \mathbb{N}^+$, if and only if, the following conditions hold:*

$$\begin{aligned} & \Theta \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \quad (5.28a) \\ \wedge \quad & \Theta \geq \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \quad (5.28b) \\ \wedge \quad & \Theta \geq \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \quad (5.28c) \end{aligned}$$

Proof: For the "only if" direction, we must show if some point of the line segment is in the ℓ-feasibility region for any given $\ell \in \mathbb{N}^+$ then the three conditions of Equation (5.28) hold. We will show this by contrapositive; that is, if any of the three conditions is violated, the line segment will not be in $\mathcal{F}_\ell(\Pi, \Delta, \Theta)$ for that ℓ. We now consider the negation of the conditions of Equation (5.28). By negation, at least one of the Equations (5.28a), (5.28b), or (5.28c) must be violated. We will show that if any of the conditions is violated, then for all $(t, D_t) \in \mathcal{L}^i_{t_a}$, $(t, D_t) \notin \mathcal{F}_\ell(\Pi, \Delta, \Theta)$.

Case 1: *Equation (5.28a) is false.* That is,

$$\begin{aligned} & \Theta \;\; < \frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell + 1} \\ \Rightarrow \quad & \Theta \;\; < \frac{\bar{D}_{t_a} + \alpha(t_{a+1} - t_a) - t_{a+1} + \ell\Pi + \Delta}{\ell + 1}. \end{aligned}$$

The second inequality follows from the fact that $D_{t_{a+1}} = \bar{D}_{t_a} + \alpha(t_{a+1} - t_a)$. Consider any $(t, D_t) \in \mathcal{L}^i_{t_a}$. Let $x \stackrel{\text{def}}{=} t - t_a$ where $0 \leq x \leq t_{a+1} - t_a$; thus, $t = t_a + x$ and $D_t = \bar{D}_{t_a} + \alpha x$. Consider the expression

$$\frac{(\bar{D}_{t_a} + \alpha x) - (t_a + x) + \ell\Pi + \Delta}{\ell + 1}$$

Obviously, the above expression is non-increasing in x, since $U_\tau \leq 1$ and α is at most the utilization of tasks with higher priority than τ_i. Therefore, $\frac{\bar{D}_{t_a}+\alpha(t_{a+1}-t_a)-t_{a+1}+\ell\Pi+\Delta}{\ell+1} \leq -\frac{(\bar{D}_{t_a}+\alpha x)-(t_a+x)+\ell\Pi+\Delta}{\ell+1} \leq \frac{D_t-t+\ell\Pi+\Delta}{\ell+1}$ for all $(t, D_t) \in \mathcal{L}^i_{t_a}$. This implies that the first condition of ℓ-feasibility is violated for all (t, D_t).

Case 2: *Equation (5.28b) is false*. That is, $\Theta < \bar{D}_{t_a}/\ell$. Again, consider any $(t, D_t) \in \mathcal{L}^i_{t_a}$. Observe that $D_t = \bar{D}_{t_a} + \alpha(t - t_a) \geq \bar{D}_{t_a}$, since $t \geq t_a$ and $\alpha \geq 0$. Thus, $\Theta < \bar{D}_{t_a}/\ell$ implies $\Theta < D_t/\ell$ for all (t, D_t); this implies that the second condition of $\mathcal{F}_\ell(\Pi, \Delta, \Theta)$ is violated.

Case 3: *Equation (5.28c) is false*. That is,

$$\Theta < \frac{\bar{D}_{t_a} + \alpha(\ell\Pi + \Delta - t_a)}{\ell + \alpha}. \tag{5.28}$$

Consider any $(t, D_t) \in \mathcal{L}^i_{t_a}$. We consider two further subcases based on the value of t. We will show in both subcases, $(t, D_t) \notin \mathcal{F}_\ell(\Pi, \Delta, \Theta)$.

Subcase 3a: $t < \frac{\bar{D}_{t_a}-\alpha t_a+\ell\Pi+\Delta-(\ell+1)\Theta}{1-\alpha}$.

By solving for Θ, we obtain

$$\begin{aligned} & \Theta < \tfrac{\bar{D}_{t_a}-(1-\alpha)t-\alpha t_a+\ell\Pi+\Delta}{\ell+1} \\ \Rightarrow \quad & \Theta < \tfrac{D_t-t+\ell\Pi+\Delta}{\ell+1} \end{aligned}$$

The implication follows from $D_t = \bar{D}_{t_a} + \alpha(t - t_a)$. The above inequality implies that the first condition of ℓ-feasibility is violated.

Subcase 3b: $t \geq \frac{\bar{D}_{t_a}-\alpha t_a+\ell\Pi+\Delta-(\ell+1)\Theta}{1-\alpha}$.

Again, solving for Θ,

$$\begin{aligned} & \Theta \geq \frac{\bar{D}_{t_a} - (1-\alpha)t - \alpha t_a + \ell\Pi + \Delta}{\ell+1} \\ \Rightarrow \quad & \Theta \geq \frac{D_t - t + \ell\Pi + \Delta}{\ell+1} \end{aligned} \tag{5.29}$$

Now consider the value of the first partial derivative of Φ_3 with respect to α; i.e., $\frac{\partial \Phi_3}{\partial \alpha}$ which is equal to

$$\frac{\ell(\ell\Pi + \Delta - t_a) - \bar{D}_{t_a}}{(\ell+\alpha)^2}.$$

Note the sign of the above partial derivative is independent of the value of α; therefore, either $\frac{\partial \Phi_3}{\partial \alpha} \leq 0$, or $\frac{\partial \Phi_3}{\partial \alpha} > 0$ for any $\alpha : 0 \leq \alpha \leq 1$; in other words, the sign remains constant for all α. If $\frac{\partial \Phi_3}{\partial \alpha} > 0$, then Φ_3 is maximized when α is as large as possible (i.e., α equals one). By Equation (5.28), this implies that $\Theta < \frac{\bar{D}_{t_a} + \ell\Pi + \Delta - t_a}{\ell+1}$ which is impossible due to Equation (5.29). Thus, $\frac{\partial \Phi_3}{\partial \alpha} \leq 0$ must be true. If the partial derivative is non-positive, then Φ_3 is maximized when α is as small as possible (i.e., α equals zero). By Equation (5.28), $\Theta < \frac{D_t}{\ell}$ which violates the second condition of ℓ-feasibility.

Thus, we have proved that if the line segment has a point in the ℓ-feasibility region, then the conditions in Equation (5.28) hold.

For the "if" direction, we need to show, if the conditions hold then there exists a point on the line segment that is included in the ℓ-feasibility region. Again, we will show by contrapositive; that is, if the line segment is completely outside the ℓ-feasibility region, then there is a condition of Equation (5.28) that is not satisfied. Assume that for all $(t, D_t) \in \mathcal{L}^i_{t_a}$ that $(t, D_t) \notin \mathcal{F}_\ell(\Pi, \Delta, \Theta)$. The previous statement implies that the first or the second condition of ℓ-feasibility must be violated for each (t, D_t). We now consider two cases based on the "location" of the left end point of the line segment $(t_a, \bar{D}_{t_a})$.

Case 1: *The second condition of ℓ-feasibility is violated for* $(t_a, \bar{D}_{t_a})$. In this case, $\Theta < \frac{\bar{D}_{t_a}}{\ell}$. Indeed, this violates the condition of Equation (5.28b).

Case 2: *The second condition of ℓ-feasibility is not violated for* $(t_a, \bar{D}_{t_a})$. In this case, $\Theta \geq \frac{\bar{D}_{t_a}}{\ell}$. We now consider two further subcases regarding the "location" of $(t_{a+1}, D_{t_{a+1}})$.

Subcase 2a: *The first condition of ℓ-feasibility is violated for the right end point of line segment* $(t_{a+1}, D_{t_{a+1}})$. In this case, $\Theta < \frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell+1}$. This clearly violates the condition of Equation (5.28a).

Subcase 2b: *The first condition of ℓ-feasibility is not violated for* $(t_{a+1}, D_{t_{a+1}})$. In this subcase, $\frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell+1} \leq \Theta$. Consider the function $\theta(t) \stackrel{\text{def}}{=} \frac{\bar{D}_{t_a} + \alpha(t - t_a) - t + \ell\Pi + \Delta}{\ell+1}$ for $t \in [t_a, t_{a+1}]$. Thus, by this subcase and $D_{t_{a+1}} = \bar{D}_{t_a} + \alpha(t_{a+1} - t_a)$ we obtain the following equation,

$$\left(\theta(t_{a+1}) \stackrel{\text{def}}{=} \frac{\bar{D}_{t_a} + \alpha(t_{a+1} - t_a) - t_{a+1} + \ell\Pi + \Delta}{\ell + 1}\right) \leq \Theta. \tag{5.30}$$

By Case 2, the second condition of ℓ-feasibility is not violated for $(t_a, \bar{D}_{t_a})$. Thus, the first condition must be; i.e.,

$$\left(\theta(t_a) \stackrel{\text{def}}{=} \frac{\bar{D}_{t_a} - t_a + \ell\Pi + \Delta}{\ell + 1}\right) > \Theta. \tag{5.31}$$

Therefore, $\Theta \in [\theta(t_{a+1}), \theta(t_a))$. Observe that $\theta(t)$ is continuous for all $t \in [t_a, t_{a+1}]$. Therefore, the Intermediate Value Theorem implies that there exists a $t' \in [t_a, t_{a+1}]$ such that $\theta(t')$ equals Θ. That is,

$$\frac{\bar{D}_{t_a} + \alpha(t' - t_a) - t' + \ell\Pi + \Delta}{\ell + 1} = \Theta. \tag{5.32}$$

By the above equality, the first condition of ℓ-feasibility is not violated for $(t', D_{t'})$; therefore, the second condition must be false:

$$\frac{\bar{D}_{t_a} + \alpha(t' - t_a)}{\ell} > \Theta. \tag{5.33}$$

Solving Equation (5.32) for t', we obtain

$$t' = \frac{\bar{D}_{t_a} - \alpha t_a + \ell\Pi + \Delta - (\ell + 1)\Theta}{1 - \alpha}.$$

Substituting the above solution to t' into Equation (5.33) and solving for Θ, we obtain

$$\Theta < \frac{\bar{D}_{t_a} - \alpha(\ell\Pi + \Delta - t_a)}{\ell + \alpha}$$

which indeed violates the condition of Equation (5.28c).

Thus, if the line segment is strictly above the ℓ-feasibility region, at least one of the three conditions is violated. ■

The following lemma formalizes the equivalence between the concept of a line segment $\mathcal{L}^i_{t_a}$ being included in some ℓ-feasibility region and the concept of a cumulative request-bound function $\widetilde{\mathsf{RBF}}$ falling below a supply-bound function sbf.

Lemma 15 *For consecutive pair of values $(t_a, t_{a+1}) \in \widehat{\mathsf{TS}}_i(\tau, k)$ and $(t, D_t) \in \mathcal{L}^i_{t_a}$ such that $t_a < t \leq t_{a+1}$, the inequality $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)$ holds, if and only if, there exists $\ell \in \mathbb{N}^+$ such that $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Delta, \Theta)$.*

Proof: For the "if" direction, we must show that if the point $(t, D_t) \in \mathcal{L}^i_{t_a}$ satisfies $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Delta, \Theta)$, then there is sufficient supply over an interval of length t to satisfy the execution of a job of τ_i and the approximated execution times of all higher-priority tasks (formally, $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)$). Observe that every point in $\mathcal{F}_\ell(\Pi, \Delta, \Theta)$ is below the sbf function (see Figure 4.1). Thus, if $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Delta, \Theta)$, then $D_t \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)$. Finally, Lemma 13 states that $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq D_t$ implying the "if" direction.

For the "only if" direction, observe that $\mathcal{L}^i_{t_a}$ and $\widetilde{\mathsf{RBF}}(\tau_i, t)$ are equivalent for $t \in (t_a, t_{a+1}]$. Thus, we must show that if line segment $\mathcal{L}^i_{t_a}$ has point (t, D_t) contained below the sbf function for Ω, then there exists an $\ell \in \mathbb{N}^+$ such that $\langle (t, D_t), \alpha \rangle \in \mathcal{F}_\ell(\Pi, \Theta, \Delta)$. Consider $\ell = \lceil \frac{D_t}{\Theta} \rceil$. The second condition of ℓ-feasibility (Equation (5.22)) is trivially satisfied for this ℓ. It also must be true that $D_t > (\ell - 1)\Theta$. Thus, (t, D_t) must be below of the line defined by $y = x - (\ell\Pi + \Delta - (\ell + 1)\Theta)$ (otherwise, (t, D_t) would be above the sbf function at t). This last constraint is equivalent to the first condition of ℓ-feasibility region. Therefore, for $\ell = \lceil \frac{D_t}{\Theta} \rceil$ we have satisfied the two conditions of Equation (5.22), implying that $(t, D_t) \in \mathcal{F}_{\lceil \frac{D_t}{\Theta} \rceil}(\Pi, \Theta, \Delta)$. ■

In the above lemma, we did not include t_a in the interval of time values where line segment inclusion in the ℓ-feasibility region implies that the approximate request-bound function is below the supply-bound

function. The exclusion of t_a from the above lemma is due to the fact that $\widetilde{\mathsf{RBF}}$ is discontinuous at t_a. However, notice that t_a is the right end point of the predecessor line segment immediately to the left of $\mathcal{L}^i_{t_a}$.

Lemma 14 equates the concept of finding t such that $\widetilde{\mathsf{RBF}}(\tau_i, t)$ is below the sbf for a given Θ and the concept of point (t, D_t) of a line segment $\mathcal{L}^i_{t_a}$ being contained in some ℓ-feasibility region for Θ. The next lemma uses Definitions 12 and 13 to show that if we can compute $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$ for any $\ell \in \mathbb{N}^+$, then we can also compute $\Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$.

Lemma 16

$$\Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a}) = \inf_{\ell > 0} \left\{ \Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a}) \right\}. \tag{5.34}$$

Proof: Let Θ_{RHS} denote the right-hand side of Equation (5.34). We will show that both $\Theta_{\mathrm{RHS}} \geq \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$ and $\Theta_{\mathrm{RHS}} \leq \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$ which will imply the lemma. First, we show $\Theta_{\mathrm{RHS}} \geq \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$. By definition of infimum, for any $\delta > 0$, there exists $\ell \in \mathbb{N}^+$ such that $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a}) \leq \Theta_{\mathrm{RHS}} + \delta$. Definition 13 states that there exists $(t, D_t) \in \mathcal{L}^i_{t_a}$ such that $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta_{\mathrm{RHS}} + \delta, \Delta)$ for this ℓ. Therefore, for all $\delta > 0$, $\Theta_{\mathrm{RHS}} + \delta$ must be in the set considered in the $\inf$ on the right-hand side of Equation (5.26) by Definition 12. Thus, $\Theta_{\mathrm{RHS}} \geq \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$.

Next, we will show $\Theta_{\mathrm{RHS}} \leq \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$. By Definition 12 and application of Lemma 13, there exist $(t, D_t) \in \mathcal{L}^i_{t_a}$ such that

$$\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Delta, \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})), t).$$

Lemma 15 implies that there exists $\ell \in \mathbb{N}^+$ such that $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a}), \Delta)$. By Definition 13, this implies that $\Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$ is in the set considered in the right-hand side of Equation (5.27) which implies the inequality. ■

In the next few lemmas, we derive the values ℓ_1 and ℓ_2 (Equations (5.24) and (5.25)), and prove that we only need to evaluate the Φ functions at these ℓ values to obtain minimum capacity. Consider the three conditions given in Equation (5.28) of Lemma 14. There are three possible cases. We invite the reader to verify that these cases are complete and mutually exclusive.

Case I: $\big(\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)\big) \wedge \big(\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)\big)$;

Case II: $\big(\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)\big) \wedge \big(\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)\big)$;

Case III: $\big(\Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)\big) \wedge \big(\Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)\big)$.

For each of the above cases, we solve for the value of ℓ and obtain bounds for the value of ℓ.

Lemma 17 *For any $\mathcal{L}^i_{t_a}$, and $\ell \in \mathbb{N}^+$, Π, Δ, Case I holds, if and only if,*

$$\ell \geq \left\lfloor \frac{(t_{a+1} - \Delta) + \sqrt{(t_{a+1} - \Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi} \right\rfloor + 1. \tag{5.35}$$

Proof: Let us consider the "only if" direction of the lemma; that is, Case I holds. From Case I, we have that both $\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$ and $\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$. For $\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$, solving for ℓ,

$$\begin{aligned} & \frac{D_{t_{a+1} - t_{a+1} + \ell\Pi + \Delta}}{\ell + 1} > \frac{\bar{D}_{t_a}}{\ell} \\ \Leftrightarrow \quad & \ell > \frac{[(1-\alpha)t_{a+1} + \alpha t_a - \Delta] + \sqrt{((1-\alpha)t_{a+1} + \alpha t_a - \Delta)^2 + 4\Pi \bar{D}_{t_a}}}{2\Pi}. \end{aligned} \tag{5.36}$$

The bidirectional implication follows since Inequality (5.36) is a quadratic inequality with respect to ℓ, defining a convex parabola $\Pi\ell^2 - ((1-\alpha)t_{a+1} + \alpha t_a - \Delta)\,\ell - \bar{D}_{t_a}$. The zeros of the parabola are

$$\frac{[(1-\alpha)t_{a+1} + \alpha t_a - \Delta] \pm \sqrt{((1-\alpha)t_{a+1} + \alpha t_a - \Delta)^2 + 4\Pi \bar{D}_{t_a}}}{2\Pi}.$$

Since the square-root term in the numerator is always greater than the term preceding the $\pm$, one root is positive and the other is negative. Inequality (5.36) implies that we are interested in values of $\ell \in \mathbb{N}^+$ such that the parabola strictly exceeds zero. Since the parabola is convex, all values of ℓ strictly greater than the positive root satisfy this inequality.

For $\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$, solving for ℓ,

$$\begin{aligned} & \frac{D_{t_{a+1} - t_{a+1} + \ell\Pi + \Delta}}{\ell + 1} > \frac{\bar{D}_{t_a} + \alpha(\ell\Pi + \Delta - t_a)}{\ell + \alpha} \\ \Leftrightarrow \quad & \ell > \frac{(t_{a+1} - \Delta) + \sqrt{(t_{a+1} - \Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi} \end{aligned} \tag{5.37}$$

The bidirectional implication follows since Inequality (5.37) is a quadratic inequality with respect to ℓ, defining a convex parabola $\Pi\ell^2 - (t_{a+1} - \Delta)\,\ell - \big(\big(\bar{D}_{t_a} + \alpha(t_{a+1} - t_a)\big)$. By similar reasoning done for

Inequality (5.36), all values of ℓ strictly greater than the positive root satisfy this inequality.

Combining Equations (5.36) and (5.37), we obtain

$$\ell > \max \left\{ \begin{array}{l} \frac{[(1-\alpha)t_{a+1}+\alpha t_a-\Delta]+\sqrt{((1-\alpha)t_{a+1}+\alpha t_a-\Delta)^2+4\Pi\bar{D}_{t_a}}}{2\Pi}, \\ \frac{(t_{a+1}-\Delta)+\sqrt{(t_{a+1}-\Delta)^2+4\Pi D_{t_{a+1}}}}{2\Pi} \end{array} \right\}. \tag{5.38}$$

Observe that $(1-\alpha)t_{a+1} + \alpha t_a - \Delta$ equals $t_{a+1} - \Delta - \alpha(t_{a+1} - t_a)$ which is at most $t_{a+1} - \Delta$, since $t_{a+1} > t_a$ and $0 \leq \alpha \leq 1$. Thus, we conclude that the second value of Equation (5.38) is the maximum of the two bounds obtained in this case. The lemma follows by observing that ℓ is an integer. The "if" direction follows by simply reversing the direction of each implication in the proof. ■

Lemma 18 *For any $\mathcal{L}^i_{t_a}$ and $\ell \in \mathbb{N}^+$, Π, Δ, Case II holds, if and only if,*

$$\ell \leq \left\lfloor \frac{(t_a-\Delta)+\sqrt{(t_a-\Delta)^2+4\Pi\bar{D}_{t_a}}}{2\Pi} \right\rfloor . \ell \leq \left\lceil \frac{(t_a-\Delta)+\sqrt{(t_a-\Delta)^2+4\Pi\bar{D}_{t_a}}}{2\Pi} \right\rceil - 1. \tag{5.39}$$

Proof: Let us consider the "only if" direction of the lemma; that is, Case II holds. From Case II, we have that both $\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$ and $\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$. For $\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) > \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$, solving for ℓ,

$$\begin{array}{rl} & \frac{\bar{D}_{t_a}}{\ell} > \frac{\bar{D}_{t_a}+\alpha(\ell\Pi+\Delta-t_a)}{\ell+\alpha} \\ \Leftrightarrow & \ell < \frac{(t_a-\Delta)+\sqrt{(t_a-\Delta)^2+4\Pi\bar{D}_{t_a}}}{2\Pi} \end{array} \tag{5.40}$$

The bidirectional implication follows since Inequality (5.40) is a quadratic inequality with respect to ℓ, defining a convex parabola $\Pi\ell^2 - (t_a - \Delta)\,\ell - \bar{D}_{t_a}$. The zeros of the parabola are

$$\frac{(t_a-\Delta) \pm \sqrt{(t_a-\Delta)^2+4\Pi\bar{D}_{t_a}}}{2\Pi}.$$

Since the square-root term in the numerator is always greater than the term preceding the $\pm$, one root is positive and the other is negative. Inequality (5.40) implies that we are interested in values of $\ell \in \mathbb{N}^+$ such that the parabola is strictly below zero. Since the parabola is convex, all positive integer values of ℓ strictly less than the positive root satisfy this inequality.

For $\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$, solving for ℓ,

$$\begin{aligned} & \frac{\bar{D}_{t_a}}{\ell} \geq \frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell+1} \\ \Leftrightarrow \quad & \ell \leq \frac{[(1-\alpha)t_{a+1} + \alpha t_a - \Delta] + \sqrt{((1-\alpha)t_{a+1} + \alpha t_a - \Delta)^2 + 4\Pi\bar{D}_{t_a}}}{2\Pi} \end{aligned} \tag{5.41}$$

The bidirectional implication follows since Inequality (5.41) is a quadratic inequality with respect to ℓ, defining a convex parabola $\Pi\ell^2 - ((1-\alpha)t_{a+1} + \alpha t_a - \Delta)\,\ell - \bar{D}_{t_a}$. By similar reasoning done for Inequality (5.40), all positive integer values of ℓ at most the positive root satisfy this inequality.

Now consider the following term: $(1-\alpha)t_{a+1} + \alpha t_a - \Delta$ which equals $(1-\alpha)(t_{a+1} - t_a) + t_a - \Delta$ which is at least $t_a - \Delta$ since $t_{a+1} > t_a$ and $0 \leq \alpha \leq 1$. Thus, we conclude that the value on the right-hand-side of Equation (5.40) is the minimum of the two values obtained in this case. The lemma follows by observing that ℓ must be an integer. The "if" direction of the lemma follows by simply reversing the implications of the proof. ■

Lemma 19 *For any $\mathcal{L}^i_{t_a}$ and $\ell \in \mathbb{N}^+$, Π, Δ, Case III holds, if and only if,*

$$\left\lceil \frac{(t_a - \Delta) + \sqrt{(t_a - \Delta)^2 + 4\Pi\bar{D}_{t_a}}}{2\Pi} \right\rceil \leq \ell \leq \left\lfloor \frac{(t_{a+1} - \Delta) + \sqrt{(t_{a+1} - \Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi} \right\rfloor \tag{5.42}$$

Proof: Let us consider the "only if" direction of the lemma; that is, Case III holds. From Case III, we have that both $\Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$ and $\Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$. For $\Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$, solving for ℓ,

$$\begin{aligned} & \frac{\bar{D}_{t_a} + \alpha(\ell\Pi + \Delta - t_a)}{\ell + \alpha} \geq \frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell+1} \\ \Leftrightarrow\ & \ell \leq \frac{(t_{a+1} - \Delta) + \sqrt{(t_{a+1} - \Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi} \end{aligned} \tag{5.43}$$

The bidirectional implication follows since Inequality (5.43) is a quadratic inequality with respect to ℓ, defining a convex parabola $\Pi\ell^2 - (t_{a+1} - \Delta)\,\ell - \big(\bar{D}_{t_a} + \alpha(t_{a+1} - t_a)\big)$. The zeros of the parabola are

$$\frac{(t_{a+1} - \Delta) \pm \sqrt{(t_{a+1} - \Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi}.$$

Since the square-root term in the numerator is always greater than the term preceding the $\pm$, one root is positive and the other is negative. Inequality (5.43) implies that we are interested in values of $\ell \in \mathbb{N}^+$ such that the parabola is at most zero. Since the parabola is convex, all positive integer values of ℓ at most the positive root satisfy this inequality.

For $\Phi_3(\mathcal{L}_{t_a}^i, \ell, \Pi, \Delta) \geq \Phi_2(\mathcal{L}_{t_a}^i, \ell, \Pi, \Delta)$, solving for ℓ,

$$\begin{aligned} &\frac{\bar{D}_{t_a}+\alpha(\ell\Pi+\Delta-t_a)}{\ell+\alpha} \geq \frac{\bar{D}_{t_a}}{\ell} \\ &\Leftrightarrow \ell \geq \frac{(t_a-\Delta)+\sqrt{(t_a-\Delta)^2+4\Pi\bar{D}_{t_a}}}{2\Pi} \end{aligned} \tag{5.44}$$

The bidirectional implication follows since Inequality (5.44) is a quadratic inequality with respect to ℓ, defining a convex parabola $\Pi\ell^2 - (t_a - \Delta)\,\ell - \bar{D}_{t_a}$. The zeros of the parabola are

$$\ell \geq \frac{(t_a - \Delta) \pm \sqrt{(t_a - \Delta)^2 + 4\Pi\bar{D}_{t_a}}}{2\Pi}.$$

Since the square-root term in the numerator is always greater than the term preceding the $\pm$, one root is positive and the other is negative. Inequality (5.44) implies that we are interested in values of $\ell \in \mathbb{N}^+$ such that the parabola is at least zero. Since the parabola is convex, all positive integer values of ℓ at least the positive root satisfy this inequality.

The lemma follows by observing that ℓ must be an integer. The "if" direction of the lemma follows by simply reversing the implications of the proof. ■

We now prove three lemmas and corollaries which show that for all $\ell \in \mathbb{N}^+$ not equal to the values $\lfloor \ell_1 \rfloor, \lfloor \ell_1 \rfloor + 1, \lceil \ell_2 \rceil$ or $\lceil \ell_2 \rceil - 1$ will result in a larger minimum Θ. The first lemma, towards this goal, shows that if a point on the line segment is in an ℓ'-feasibility region and ℓ' is at least $\lfloor \ell_1 \rfloor + 1$, then the point is also in the $\lfloor \ell_1 \rfloor + 1$-feasibility region.

Lemma 20 *For any $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $(t, D_t) \in \mathcal{L}_{t_a}^i$, $\ell' \in \mathbb{N}^+$, Π, Δ, and Θ, if $\ell' \geq \lfloor \ell_1 \rfloor + 1$ and $\Theta \leq \Delta$ then*

$$[(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)] \Rightarrow \left[(t, D_t) \in \mathcal{F}_{\lfloor \ell_1 \rfloor + 1}(\Pi, \Delta, \Theta)\right].$$

Proof: By Lemma 17 and $\ell' \geq \lfloor \ell_1 \rfloor + 1$, Case I must hold for all such ℓ'. Combining Case I and Lemma 14, we have that if $(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)$, then

$$\Theta \geq \Phi_1(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta).$$

Now consider the first partial derivative of Φ_1 with respect to ℓ; i.e.,

$$\begin{aligned} \frac{\partial \Phi_1}{\partial \ell} &= \frac{-D_{t_{a+1}} + t_{a+1} - \ell\Pi - \Delta + \Pi(\ell+1)}{(\ell+1)^2} \\ &= \frac{[t_{a+1} - D_{t_{a+1}}] + [\Pi - \Delta]}{(\ell+1)^2}. \end{aligned}$$

Since $\Pi \geq \Delta$, the second term in the numerator is positive. Consider the first term, $t_{a+1} - D_{t_{a+1}}$. By $(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)$ and the first condition of ℓ'-feasibility,

$$\begin{aligned} & t \geq D_t + \ell\Pi + \Delta - (\ell+1)\Theta \\ \Rightarrow \quad & t + (t_{a+1} - t) \geq D_t + \alpha(t_{a+1} - t) + \ell\Pi + \Delta - (\ell+1)\Theta \\ & \qquad \text{(since } \alpha < 1\text{)} \\ \Rightarrow \quad & t_{a+1} \geq D_{t_{a+1}} + \ell\Pi + \Delta - (\ell+1)\Theta \\ \Rightarrow \quad & t_{a+1} \geq D_{t_{a+1}}. \end{aligned}$$

The second to last implication is due to $D_t = \bar{D}_{t_a} + \alpha(t - t_a)$ and $D_{t_{a+1}} = \bar{D}_{t_a} + \alpha(t_{a+1} - t_a)$. The last implication is due to $\Theta \leq \Delta$. Therefore, the first term in the numerator of $\frac{\partial \Phi_1}{\partial \ell}$ is also positive. Thus, $\frac{\partial \Phi_1}{\partial \ell}$ is non-decreasing for all ℓ'. Thus, the Φ_1 evaluated at $\lfloor \ell_1 \rfloor + 1$ is a lower bound; i.e., for all $\ell' \geq \lfloor \ell_1 \rfloor + 1$,

$$\Phi_1(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta) \geq \Phi_1(\mathcal{L}^i_{t_a}, \lfloor \ell_1 \rfloor + 1, \Pi, \Delta).$$

The above inequality implies that $\Theta \geq \Phi_1(\mathcal{L}^i_{t_a}, \lfloor \ell_1 \rfloor + 1, \Pi, \Delta)$, satisfying Equation (5.28a) of Lemma 14. For $\lfloor \ell_1 \rfloor + 1$, Case I holds, implying that Equations (5.28b) and (5.28c) must also hold. Thus, by Lemma 14, $(t, D_t) \in \mathcal{F}_{\lfloor \ell_1 \rfloor + 1}(\Pi, \Delta, \Theta)$. ■

The next corollary follows from the above lemma and the definition of Θ^*_ℓ (Definition 13).

Corollary 7 *For any $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $\ell' \in \mathbb{N}^+$, Π, and Δ, if $(\ell' \geq \lfloor \ell_1 \rfloor + 1)$ then*

$$\Theta^*_{\ell'}(\Pi, \Delta, \mathcal{L}^i_{t_a}) \geq \Theta^*_{\lfloor \ell_1 \rfloor + 1}(\Pi, \Delta, \mathcal{L}^i_{t_a}).$$

The next lemma shows that if a point on the line segment is in an ℓ'-feasibility region and ℓ' is at most $\lceil \ell_2 \rceil - 1$, then the point is also in the $\lceil \ell_2 \rceil - 1$-feasibility region.

Lemma 21 *For any* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $(t, D_t) \in \mathcal{L}^i_{t_a}$, $\ell' \in \mathbb{N}^+$, Π, Δ, *and* Θ, *if* $\ell' \leq \lceil \ell_2 \rceil - 1$ *and* $\Theta \leq \Delta$ *then*

$$[(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)] \Rightarrow \left[(t, D_t) \in \mathcal{F}_{\lceil \ell_2 \rceil - 1}(\Pi, \Delta, \Theta)\right].$$

Proof:

By Lemma 18 and $\ell' \leq \lceil \ell_2 \rceil - 1$, Case II must hold for all such ℓ'. Combining Case II and Lemma 14, we have that if $(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)$, then

$$\Theta \geq \Phi_2(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta).$$

Now consider the first partial derivative of Φ_2 with respect to ℓ;

$$\frac{\partial \Phi_2}{\partial \ell} = \frac{-D_t}{\ell^2}.$$

Therefore, Φ_2 is a decreasing function for all $\ell' \in \mathbb{N}^+$ such that $\ell' \leq \lceil \ell_2 \rceil - 1$. Thus, the Φ_2 evaluated at $\lceil \ell_2 \rceil - 1$ is an upper bound for all such ℓ'; i.e., for all $\ell' \leq \lceil \ell_2 \rceil - 1$,

$$\Phi_2(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta) \leq \Phi_2(\mathcal{L}^i_{t_a}, \lceil \ell_2 \rceil - 1, \Pi, \Delta).$$

The above inequality implies that $\Theta \geq \Phi_2(\mathcal{L}^i_{t_a}, \lceil \ell_2 \rceil - 1, \Pi, \Delta)$, satisfying Equation (5.28b) of Lemma 14. For $\lceil \ell_2 \rceil - 1$, Case II holds, implying that Equations (5.28a) and (5.28c) must also hold. Thus, by Lemma 14, $(t, D_t) \in \mathcal{F}_{\lceil \ell_2 \rceil - 1}(\Pi, \Delta, \Theta)$. ■

The next corollary follows from the above lemma and the definition of Θ^*_ℓ (Definition 13).

Corollary 8 *For any* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $\ell' \in \mathbb{N}^+$, Π, *and* Δ, *if* $(\ell' \leq \lceil \ell_2 \rceil - 1)$ *then*

$$\Theta^*_{\ell'}(\Pi, \Delta, \mathcal{L}^i_{t_a}) \geq \Theta^*_{\lceil \ell_2 \rceil - 1}(\Pi, \Delta, \mathcal{L}^i_{t_a}).$$

Lemma 22 *For any* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $(t, D_t) \in \mathcal{L}^i_{t_a}$, $\ell' \in \mathbb{N}^+$, Π, Δ, *and* Θ, *if* $\lceil \ell_2 \rceil \leq \ell' \leq \lfloor \ell_1 \rfloor$ *then*

$$[(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)] \Rightarrow \left[(t, D_t) \in \mathcal{F}_{\lfloor \ell_1 \rfloor}(\Pi, \Delta, \Theta)\right] \vee \left[(t, D_t) \in \mathcal{F}_{\lceil \ell_2 \rceil}(\Pi, \Delta, \Theta)\right].$$

Proof: By Lemma 19 and $\lceil \ell_2 \rceil \leq \ell' \leq \lfloor \ell_1 \rfloor$, Case III must hold for all such ℓ'. Combining Case III and Lemma 14, we have that if $(t, D_t) \in \mathcal{F}_{\ell'}(\Pi, \Delta, \Theta)$, then

$$\Theta \geq \Phi_3(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta).$$

Now consider the first partial derivative of Φ_3 with respect to ℓ;

$$\frac{\partial \Phi_3}{\partial \ell} = \frac{\alpha^2 \Pi - \bar{D}_{t_a} - \alpha \Delta + \alpha t_a}{(\ell + \alpha)^2}.$$

Note the sign of the above partial derivative is independent of the value of ℓ; therefore, either $\frac{\partial \Phi_3}{\partial \ell} \leq 0$, or $\frac{\partial \Phi_3}{\partial \ell} > 0$ for any $\ell \in \mathbb{N}^+$; in other words, the sign remains constant for all ℓ. If $\frac{\partial \Phi_3}{\partial \ell} > 0$, then Φ_3 is minimized when ℓ is as small as possible; i.e., ℓ equals $\lceil \ell_2 \rceil$. In this case, the Φ_3 evaluated at $\lceil \ell_2 \rceil$ is a lower bound for all such ℓ'; i.e., for all ℓ' such that $\lceil \ell_2 \rceil \leq \ell' \leq \lfloor \ell_1 \rfloor$,

$$\Phi_3(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta) \geq \Phi_3(\mathcal{L}^i_{t_a}, \lceil \ell_2 \rceil, \Pi, \Delta).$$

The above inequality implies that $\Theta \geq \Phi_3(\mathcal{L}^i_{t_a}, \lceil \ell_2 \rceil, \Pi, \Delta)$, satisfying Equation (5.28c) of Lemma 14. For $\lceil \ell_2 \rceil$, Case III holds, implying that Equations (5.28a) and (5.28b) must also hold. Thus, by Lemma 14, $(t, D_t) \in \mathcal{F}_{\lceil \ell_2 \rceil}(\Pi, \Delta, \Theta)$ when $\frac{\partial \Phi_3}{\partial \ell} > 0$.

If $\frac{\partial \Phi_3}{\partial \ell} \leq 0$, then Φ_3 is minimized when ℓ is as large as possible; i.e., ℓ equals $\lfloor \ell_1 \rfloor$. In this case, the Φ_3 evaluated at $\lfloor \ell_1 \rfloor$ is an upper bound for all such ℓ'; i.e., for all ℓ' such that $\lceil \ell_2 \rceil \leq \ell' \leq \lfloor \ell_1 \rfloor$,

$$\Phi_3(\mathcal{L}^i_{t_a}, \ell', \Pi, \Delta) \geq \Phi_3(\mathcal{L}^i_{t_a}, \lfloor \ell_1 \rfloor, \Pi, \Delta).$$

By the same argument for $\frac{\partial \Phi_3}{\partial \ell} > 0$, $(t, D_t) \in \mathcal{F}_{\lfloor \ell_1 \rfloor}(\Pi, \Delta, \Theta)$ when $\frac{\partial \Phi_3}{\partial \ell} \leq 0$. ■

The next corollary follows from the above lemma and the definition of Θ^*_ℓ (Definition 13).

Corollary 9 *For any* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $\ell' \in \mathbb{N}^+$, Π, *and* Δ, *if* $(\lceil \ell_2 \rceil \leq \ell' \leq \lfloor \ell_1 \rfloor)$ *then*

$$\Theta^*_{\ell'}(\Pi, \Delta, \mathcal{L}^i_{t_a}) \leq \min\{\Theta^*_{\lfloor \ell_1 \rfloor}(\Pi, \Delta, \mathcal{L}^i_{t_a}), \Theta^*_{\lceil \ell_2 \rceil}(\Pi, \Delta, \mathcal{L}^i_{t_a})\}.$$

Combining Corollaries 7, 8, 9, and using Definitions 12 and 13, we obtain the following corollary.

Corollary 10

$$\Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a}) = \min_{\ell \in \{\lfloor \ell_1 \rfloor, \lfloor \ell_1 \rfloor + 1, \lceil \ell_2 \rceil, \lceil \ell_2 \rceil - 1\}} \{\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})\}.$$

By the above corollary, we now know how to compute $\Theta^*(\cdot)$ efficiently from $\Theta^*_\ell(\cdot)$. The next lemma shows that we may use the Φ functions to efficiently compute $\Theta^*_\ell(\cdot)$.

Lemma 23 *For any* $\mathcal{L}^i_{t_a}$ *and* $\ell \in \mathbb{N}^+$,

$$\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a}) = \begin{cases} \Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta), & \textit{if } \ell \geq \lfloor \ell_1 \rfloor + 1; \\ \Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta), & \textit{if } \ell \leq \lceil \ell_2 \rceil - 1; \\ \Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta), & \textit{otherwise.} \end{cases} \tag{5.45}$$

Proof: From Definition 13, $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$ is the minimum $\Theta \leq \Delta$ such that there exists $(t, D_t) \in \mathcal{L}^i_{t_a}$ where $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta, \Delta)$. By Lemma 14, such a Θ is also the minimum value that satisfied all three conditions of Equation (5.28). Since each of the conditions is a lower bound on Θ (with equality permitted), Θ must satisfy equality of at least one of the three conditions of Equation (5.28) and must exceed or equal the other two conditions. Notice that, if $\ell \geq \lfloor \ell_1 \rfloor + 1$, then by Lemma 17, Θ equals $\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$. We can show an identical proof for intervals $(0, \lceil \ell_2 \rceil - 1]$ and$[\lceil \ell_2 \rceil, \lfloor \ell_1 \rfloor]$, by applying Lemmas 18 and 19, respectively. ■

The final lemma that we prove before providing a proof for Theorem 7 shows that a choice of Θ based on the computation of $\Theta^*(\cdot)$ is a "safe" choice in the sense that all tasks in τ_i will complete by their deadline under an EDP resource $\Omega = (\Pi, \Theta, \Delta)$.

Lemma 24 *For all* $\tau_i \in \tau$, $\exists t \in (0, d_i]$ *such that* $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta, \Delta), t)$ *and* $U_\tau \leq \frac{\Theta}{\Pi}$, *if and only if,*

$$\Theta \geq \max \left(\begin{array}{l} \max_{\tau_i \in \tau} \left\{ \min_{t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i} \left\{ \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a}) \right\} \right\}, \\ U_\tau \cdot \Pi \end{array} \right). \tag{5.46}$$

Proof: We will prove this lemma by contrapositive. For the "if" direction, we must prove if either $U_\tau > \frac{\Theta}{\Pi}$ or $\forall t \in (0, d_i] : \widetilde{\mathsf{RBF}}(\tau_i, t) > \mathsf{sbf}((\Pi, \Theta, \Delta), t)$, then the negation of the inequality of Equation (5.46) is true. If we consider $U_\tau > \frac{\Theta}{\Pi}$, the inequality of Equation (5.46) is trivially violated due to the second expression in the outer $\max$ of Equation (5.46).

Now, consider the case when there exists a $\tau_i \in \tau$ such that $\widetilde{\mathsf{RBF}}(\tau_i, t) > \mathsf{sbf}((\Pi, \Theta, \Delta), t)$ for all t in $(0, d_i]$. By Lemma 15, this implies for all $\ell \in \mathbb{N}^+$, $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, and $(t, D_t) \in \mathcal{L}^i_{t_a}$ that $(t, D_t) \notin \mathcal{F}_\ell(\Pi, \Delta, \Theta)$. By Definition 13, it must be for all $\ell \in \mathbb{N}^+$ that $\Theta < \Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$. By Lemma 16, this implies that $\Theta < \Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$ for any $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, which violates the inequality of Equation (5.46) due to the first term in the outer $\max$. For the "only if" direction of the lemma, we will also consider the contrapositive. The contrapositive will follow by simply reversing the implications of the proof for the "if" direction. ■

After proving the above conditions, we are ready to prove Theorem 7 which states that FPMINIMUMCAPACITY returns a valid value for finite k and an exact value for $k = \infty$.

Proof of Theorem 7 We will show that $\Theta^{\min}$ returned from FPMINIMUMCAPACITY corresponds to the value on the right-hand side of Equation (5.46) of Lemma 24. The loop from Line 4 to 25 iterates through each consecutive pair of values t_a and t_{a+1} in $\widehat{\mathsf{TS}}_i(\tau, k)$ to find optimal capacity for each line segment defined by the endpoints $(t_a, \bar{D}_{t_a})$ and $(t_{a+1}, D_{t_{a+1}})$. It sets variables corresponding to $\widetilde{\mathsf{RBF}}(t_a)$ and $\widetilde{\mathsf{RBF}}(t_{a+1})$ in Lines 5 and 6 respectively. Then, in the next few lines it sets four different values to ℓ (based on ℓ_1 and ℓ_2, defined in Equations (5.24) and (5.25)) and evaluates $\Phi_j(\cdot)$ according to Lemma 23 to compute $\Theta^*_\ell(\cdot)$ for each of the four integer values of ℓ. Therefore, $\Theta^{\min}_{t_a}$, set in Line 14, equals $\Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})$ by Lemma 16. At the end of this loop it sets $\Theta^{\min}_i$ to be the minimum of $\Theta^{\min}_{t_a}$ and $\Theta^{\min}_i$ (Line 15). Thus, once the inner loop is executed for all $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, $\Theta^{\min}_i$ contains the minimum of all $\Theta^{\min}_{t_a}$ values. The outer loop from Line 1 to Line 28 finds $\Theta^{\min}_i$ for all task τ_i in τ. Finally, in Line 17, $\Theta^{\min}$ is set to the maximum of $U_\tau \cdot \Pi$ and $\Theta^{\min}_i$ over all values τ_i in τ.

By Lemma 24, $\widetilde{\mathsf{RBF}}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta^{\min}, \Delta), t)$ for some $t \in (0, d_i]$ and $U_\tau \leq \frac{\Theta}{\Pi}$. By Lemma 12, $\mathsf{RBF}(\tau_i, t) \leq \widetilde{\mathsf{RBF}}(\tau_i, t)$. This implies $\mathsf{RBF}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta^{\min}, \Delta), t)$ which is the schedulability condition given by Theorem 6. Therefore, τ will always meet all deadlines when scheduled by fixed-priority scheduling upon $\Omega = (\Pi, \Theta^{\min}, \Delta)$. When $k = \infty$, $\widetilde{\mathsf{RBF}}(\tau_i, t)$ equals $\mathsf{RBF}(\tau_i, t)$ for all $t \geq 0$; in this case, $\Theta^{\min}$ equals $\Theta^*(\Pi, \Delta, \tau)$ (i.e., $\Theta^{\min}$ is exact capacity). ■

Approximation Ratio

In the previous section, we have shown that FPMINIMUMCAPACITY gives a valid answer when k is finite and an exact answer when k is infinite. In this section, we show that as k increases, the guaranteed accuracy of FPMINIMUMCAPACITY increases along with its running time. Theorem 8 presents the tradeoff between accuracy and computational complexity, in terms of k.

Theorem 8 *Given* Π, Δ, τ, *and* $k \in \mathbb{N}^+$, *the procedure* FPMINIMUMCAPACITY *returns* $\Theta^{\min}$ *such that*

$$\Theta^*(\Pi, \Delta, \tau) \leq \Theta^{\min} \leq \left(\frac{k+1}{k}\right) \cdot \Theta^*(\Pi, \Delta, \tau).$$

Furthermore, FPMINIMUMCAPACITY (Π, Δ, τ, k) *has time complexity* $O(kn^2 \log n)$

The following corollary quantifying our FPTAS is immediately obtainable from Theorem 8, by substituting a value for k dependent on the accuracy parameter ϵ ($k = \lceil \frac{1}{\epsilon} \rceil$).

Corollary 11 *Given* Π, Δ, τ, *and* $\epsilon > 0$, *the procedure* FPMINIMUMCAPACITY $\left(\Pi, \Delta, \tau, \lceil \frac{1}{\epsilon} \rceil\right)$ *returns* $\Theta^{\min}$ *such that*

$$\Theta^*(\Pi, \Delta, \tau) \leq \Theta^{\min} \leq (1+\epsilon) \cdot \Theta^*(\Pi, \Delta, \tau).$$

Furthermore, FPMINIMUMCAPACITY $\left(\Pi, \Delta, \tau, \lceil \frac{1}{\epsilon} \rceil\right)$ *has time complexity* $O\left(\frac{n^2 \log n}{\epsilon}\right)$.

To prove Theorem 8, we need to prove two additional lemmas.

Lemma 25 *Given* Π, Δ, *and pair of consecutive pair of values* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$, *the following is true for all* $k, \ell (\in \mathbb{N}^+)$, *and* $\alpha (\in [0,1])$,

$$\Theta^*_\ell\left(\Pi, \Delta, \mathcal{L}^i_{t_a}\right) \leq \left(\tfrac{k+1}{k}\right) \cdot \Theta^*_\ell\left(\Pi, \Delta, \left\langle \left(t_a, \tfrac{k \cdot \bar{D}_{t_a}}{k+1}\right), \left(t_{a+1}, \tfrac{k \cdot D_{t_{a+1}}}{k+1}\right), \tfrac{k \cdot \alpha}{k+1} \right\rangle\right). \tag{5.47}$$

Proof: By Lemma 23, $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$ must be equal to one of Φ_1, Φ_2 or Φ_3 according to the value of ℓ. We will show that for each of the three possibilities, Equation (5.47) must hold.

If $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$ is equal to $\Phi_1(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$ (i.e., $\frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell+1}$), then $\ell \geq \lfloor \ell_1 \rfloor + 1$ by Lemma 23. This implies by definition of ℓ_1,

$$\begin{aligned}
\ell &\geq \left\lfloor \frac{(t_{a+1}-\Delta) + \sqrt{(t_{a+1}-\Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi} \right\rfloor + 1 \\
&> \frac{(t_{a+1}-\Delta) + \sqrt{(t_{a+1}-\Delta)^2 + 4\Pi D_{t_{a+1}}}}{2\Pi} \\
&> \frac{2(t_{a+1}-\Delta)}{2\Pi} \\
&= \frac{t_{a+1}-\Delta}{\Pi}.
\end{aligned}$$

Thus, $\ell\Pi + \Delta - t_{a+1} \geq 0$. By Lemma 23 and $\ell \geq \lfloor \ell_1 \rfloor + 1$,

$$\begin{aligned}
&\Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \bar{D}_{t_a}}{k+1} \right), \left(t_{a+1}, \frac{k \cdot D_{t_{a+1}}}{k+1} \right), \frac{k \cdot \alpha}{k+1} \right\rangle \right) \\
&= \Phi_1 \left(\left\langle \left(t_a, \tfrac{k \cdot \bar{D}_{t_a}}{k+1} \right), \left(t_{a+1}, \tfrac{k \cdot D_{t_{a+1}}}{k+1} \right), \tfrac{k \cdot \alpha}{k+1} \right\rangle, \ell, \Pi, \Delta \right) \\
&= \frac{\frac{k \cdot D_{t_{a+1}}}{k+1} - t_{a+1} + \ell\Pi + \Delta}{\ell+1} \\
&\geq \frac{\frac{k}{k+1} \cdot D_{t_{a+1}} + \frac{k}{k+1} \cdot (\ell\Pi + \Delta - t_{a+1})}{\ell+1} \\
&\geq \left(\tfrac{k}{k+1} \right) \cdot \left(\frac{D_{t_{a+1}} - t_{a+1} + \ell\Pi + \Delta}{\ell+1} \right) \\
&= \left(\tfrac{k}{k+1} \right) \cdot \Theta^*_\ell \left(\Pi, \Delta, \mathcal{L}^i_{t_a} \right).
\end{aligned}$$

In this case, Equation (5.47) holds.

If $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$ is equal to $\Phi_2(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$ (i.e, $\frac{\bar{D}_{t_a}}{\ell}$), then $\ell \leq \lceil \ell_2 \rceil - 1$ by Lemma 23. Lemma 23 also implies

$$\begin{aligned}
&\Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \bar{D}_{t_a}}{k+1} \right), \left(t_{a+1}, \frac{k \cdot D_{t_{a+1}}}{k+1} \right), \frac{k \cdot \alpha}{k+1} \right\rangle \right) \\
&= \Phi_2 \left(\left\langle \left(t_a, \tfrac{k \cdot \bar{D}_{t_a}}{k+1} \right), \left(t_{a+1}, \tfrac{k \cdot D_{t_{a+1}}}{k+1} \right), \tfrac{k \cdot \alpha}{k+1} \right\rangle, \ell, \Pi, \Delta \right) \\
&\geq \frac{\frac{k \cdot \bar{D}_{t_a}}{k+1}}{\ell} \\
&= \left(\tfrac{k}{k+1} \right) \cdot \Theta^*_\ell \left(\Pi, \Delta, \mathcal{L}^i_{t_a} \right).
\end{aligned}$$

Finally, if $\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})$ is equal to $\Phi_3(\mathcal{L}^i_{t_a}, \ell, \Pi, \Delta)$ (i.e., $\frac{\bar{D}_{t_a}+\alpha(\ell\Pi+\Delta-t_a)}{\ell+\alpha}$),then $\lceil \ell_2 \rceil \leq \ell \leq \lfloor \ell_1 \rfloor$ by Lemma 23. Lemma 23 also implies that

$$\begin{aligned}
&\Theta^*_\ell\left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \bar{D}_{t_a}}{k+1}\right), \left(t_{a+1}, \frac{k \cdot D_{t_{a+1}}}{k+1}\right), \frac{k \cdot \alpha}{k+1} \right\rangle\right) \\
&= \Phi_3\left(\left\langle \left(t_a, \tfrac{k \cdot \bar{D}_{t_a}}{k+1}\right), \left(t_{a+1}, \tfrac{k \cdot D_{t_{a+1}}}{k+1}\right), \tfrac{k \cdot \alpha}{k+1} \right\rangle, \ell, \Pi, \Delta\right) \\
&\geq \frac{\frac{k \cdot \bar{D}_{t_a}}{k+1} - \left(\frac{k \cdot \alpha}{k+1}\right)(\ell\Pi+\Delta-t_a)}{\ell + \left(\frac{k \cdot \alpha}{k+1}\right)} \\
&\geq \left(\frac{k}{k+1}\right) \cdot \left(\frac{\bar{D}_{t_a} - \alpha(\ell\Pi+\Delta-t_a)}{\ell+\alpha}\right) \\
&= \left(\frac{k}{k+1}\right) \cdot \Theta^*_\ell\left(\Pi, \Delta, \mathcal{L}^i_{t_a}\right).
\end{aligned}$$

■

Lemma 26 *Given* Π, Δ, $\tau_i \in \tau$, *and* $k \in \mathbb{N}^+$, *there exists consecutive pair of values* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$ *such that,*

$$\Theta^*(\Pi, \Delta, \tau) \geq \Theta^*\left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \bar{D}_{t_a}}{k+1}\right), \left(t_{a+1}, \frac{k \cdot D_{t_{a+1}}}{k+1}\right), \frac{k \cdot \alpha}{k+1} \right\rangle\right). \tag{5.48}$$

Proof:

Let Θ_{RHS} denote the right-hand side of Equation (5.48). By definition of $\Theta^*(\Pi, \Delta, \tau)$ and Theorem 6, for all $\tau_i \in \tau$, there exist $t \in (0, d_i]$ such that

$$\mathsf{RBF}(\tau_i, t) \leq \mathsf{sbf}((\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta), t). \tag{5.49}$$

Now consider any pair of consecutive values $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$. By Lemma 12, we have, for all $t \in (t_a, t_{a+1}]$,

$$\begin{aligned}
&\left(\frac{k+1}{k}\right) \cdot \mathsf{RBF}(\tau_i, t) \\
&= \left(\tfrac{k+1}{k}\right) \cdot \left(e_i + \textstyle\sum_{j=1}^{i-1} \mathsf{rbf}(\tau_j, t)\right) \\
&\geq e_i + \left(\tfrac{k+1}{k}\right) \cdot \textstyle\sum_{j=1}^{i-1} \mathsf{rbf}(\tau_j, t) \\
&\geq e_i + \left(\tfrac{k+1}{k}\right) \cdot \left(\textstyle\sum_{j=1}^{i-1} \delta(\tau_i, t) \cdot \tfrac{k}{k+1}\right) \\
&= \widetilde{\mathsf{RBF}}(\tau_i, t)
\end{aligned} \tag{5.50}$$

Combining the inequalities of Equations (5.49) and (5.50) gives us, for all $t \in (t_a, t_{a+1}]$,

$$\mathsf{sbf}((\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta), t) \geq \frac{k}{k+1} \cdot \widetilde{\mathsf{RBF}}(\tau_i, t). \tag{5.51}$$

Lemma 15 and Equation (5.51) imply that there exists $\ell \in \mathbb{N}$ and $(t, D_t) \in \left\langle \left(t_a, \frac{k \cdot \bar{D}_{t_a}}{k+1}\right), \left(t_{a+1}, \frac{k \cdot D_{t_{a+1}}}{k+1}\right), \frac{k \cdot \alpha}{k+1} \right\rangle$ such that

$$(t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta^*(\Pi, \Delta, \tau), \Delta).$$

The above expression and Definition 13 implies

$$\Theta^*_\ell \left(\Pi, \Delta, \left\langle \left(t_a, \frac{k \cdot \bar{D}_{t_a}}{k+1}\right), \left(t_{a+1}, \frac{k \cdot D_{t_{a+1}}}{k+1}\right), \frac{k \cdot \alpha}{k+1} \right\rangle\right) \leq \Theta^*(\Pi, \Delta, \tau).$$

The lemma follows from the expression above and Lemma 16. ■

We find the following corollary by combining Lemmas 25, 26 and 16.

Corollary 12 *Given* Π, Δ, $k \in \mathbb{N}^+$, *and* τ_i, *there exists consecutive pair of values* $t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i(\tau, k)$,

$$\left(\frac{k+1}{k}\right) \cdot \Theta^*(\Pi, \Delta, \tau) \geq \inf_{\ell \in \mathbb{N}^+} \left\{\Theta^*_\ell(\Pi, \Delta, \mathcal{L}^i_{t_a})\right\}. \tag{5.52}$$

Now, we are ready to give the proof of Theorem 8.

Proof of Theorem 8 We already proved the first part in Theorem 7; now we must prove the second part of the inequality. From our algorithm, the value of $\Theta^{\min}$ can be either equal to $\Pi \cdot U_\tau$ or greater than this term. If $\Theta^{\min} = \Pi \cdot U_\tau$, Theorem 6 implies that $\Theta^*(\Pi, \Delta, \tau)$ must be at least $U_\tau \cdot \Pi$. For this case, the second inequality follows, since $\frac{k+1}{k} \geq 1$ for all $k \in \mathbb{N}^+$. Now consider the case when $\Theta^{\min} > \Pi \cdot U_\tau$.

$$\Theta^{\min} = \max_{\tau_i \in \tau} \left\{ \min_{t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i} \left\{\Theta^*(\Pi, \Delta, \mathcal{L}^i_{t_a})\right\} \right\}$$

according to Theorem 7 and Lemma 24. By Lemma 16, this is equivalent to

$$\Theta^{\min} = \max_{\tau_i \in \tau} \left\{ \min_{t_a, t_{a+1} \in \widehat{\mathsf{TS}}_i} \left\{ \inf_{\ell \in \mathbb{N}^+} \left\{\Theta^*_\ell\left(\Pi, \Delta, \mathcal{L}^i_{t_a}\right)\right\} \right\} \right\}.$$

Applying Corollary 12, we find,

$$\Theta^{\min} \leq \max_{\tau_i \in \tau} \left\{ \left(\frac{k+1}{k} \right) \cdot \Theta^*(\Pi, \Delta, \tau) \right\}.$$

From this and the definition of $\Theta^*(\Pi, \Delta, \tau)$ the second inequality of this theorem follows. ■

5.3.3 Simulation Results

In this section, we present simulation results and compare the performance of our proposed algorithms. We implemented six schedulability tests: exact test derived in Section 5.2 without any heuristics (i.e., iterative convergence to determine response time in Equation 5.4); exact test with heuristics (using response time lower bound and upper bound derived in Section 5.2.1 and 5.2.2); exact algorithm by [39][3]; our proposed approximate algorithm FP-MINIMUMCAPACITY; iterative convergence-based approximate test with heuristics and sufficient algorithm by [83]. We denote these algorithms as BS-E, BS-E-h, MC-E, MC-A, BS-A-h and Suff respectively in the plots. The simulation parameters and value ranges are shown below:

1. The number of tasks in a task system τ is $[4, 60]$ at 4-increments.
2. The system utilization U_τ is taken from the range $[0.1, 0.9]$ at 0.05-increments and individual task utilizations u_i are generated using UUniFast algorithm [25].
3. Each sporadic task $\tau_i = (e_i, d_i, p_i)$ has a period p_i uniformly drawn from the interval $[10, 10000]$. The execution time e_i is set to $u_i.p_i$. We assume $d_i \leq p_i$ and is uniformly drawn from the interval $[\lceil e_i \rceil, p_i]$.
4. The component level scheduling algorithm is FP.
5. k is taken from the range $[1, 25]$ at 2-increments. Π is set in the range $[10, 10000]$; Δ is equal to Π.
6. Note that, we assume integral values for p_i, d_i, Π, Δ and fractional values for e_i and Θ for the ease of simulation, our results will still hold for non-integral parameters.
7. A 2.33 GHz Intel Core 2 Duo E6550 machine with 2.0GB RAM is used for simulations.

[3]Note that we can obtain the exact capacity from FP-MINIMUMCAPACITY with $k = \infty$.

8. Each point in the plots represents mean of 1000 simulation runs with 95% confidence intervals.

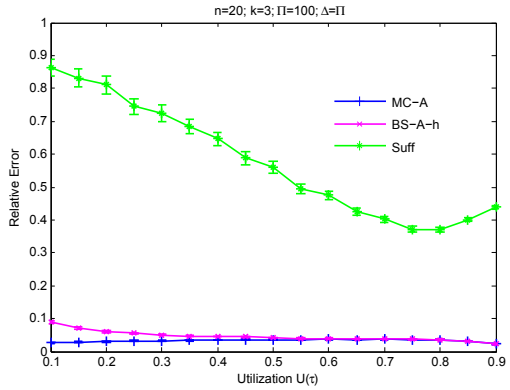


Figure 5.2: Relative Error vs System Utilization

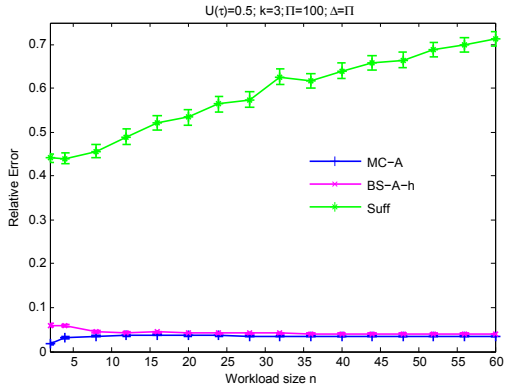


Figure 5.3: Relative Error vs Workload Size

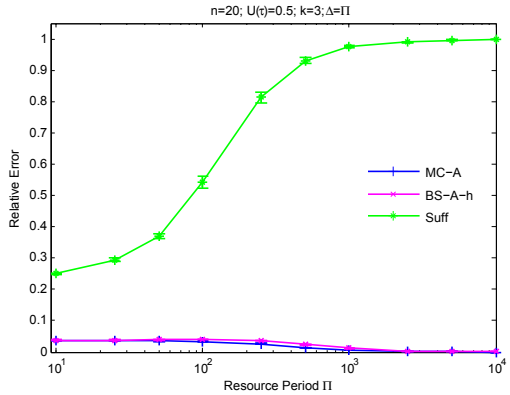


Figure 5.4: Relative Error vs Resource Period

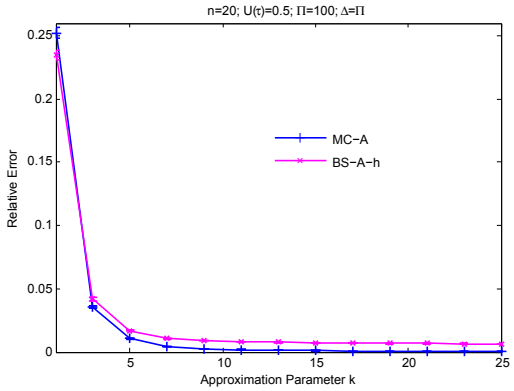


Figure 5.5: Relative Error vs Approximation Parameter

For each simulation, for a specific task system size n and utilization U_τ, we randomly generate taskset parameters u_i, p_i, e_i and d_i for each task τ_i. We execute three exact algorithms, two approximate algorithms, and the sufficient algorithm to generate exact, approximate and sufficient capacity, respectively. We first compare the relative error[4] of our proposed approximate algorithm (MC-A) with the sufficient algorithm (Suff), and iterative approximate algorithm (BS-A-h), with respect to the exact algorithm (MC-E)[5]. In Figure 5.2, the relative error in the calculation of capacity for our algorithm is plotted as a function

[4]Relative error is defined as follows: $\frac{\Theta - \Theta^*}{\Theta^*}$ where Θ^* is the exact capacity and Θ is either the sufficient capacity $\bar{\Theta}$ or the approximate capacity $\widehat{\Theta}$.

[5]Note that the relative error of BS-E and BS-E-h are equal to the threshold (equals 10^{-6} in the simulations) of the binary search used to determine minimum capacity.

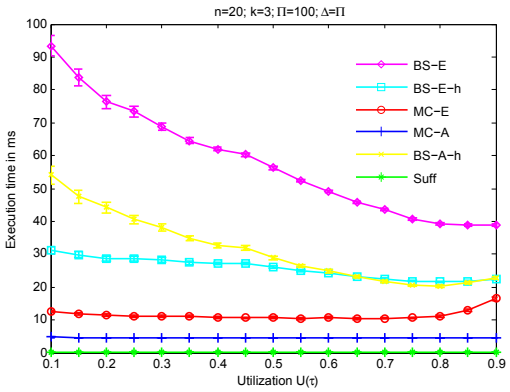


Figure 5.6: Execution Time vs System Utilization

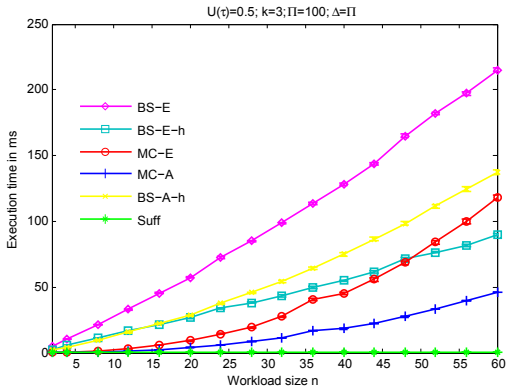


Figure 5.7: Execution Time vs Workload Size

of task system utilization ($n = 20; \Pi = 100; \Delta = \Pi; k = 3$). For MC-A, the mean relative error is less than 5%, whereas for Suff it ranges from 40% to 85%. For the sufficient algorithm, relative error is very high due to the fact that the algorithm overestimates capacity. The relative error for our approximation algorithm does not vary much with the increase in system utilization, where as it decreases for the sufficient algorithm. A potential explanation for the this is that some of the functions of Suff for setting the capacity do not depend on the utilization, only the task and resource period parameters. Such functions will be constant over increasing utilization while the optimal capacity must increase as utilization increases. This results in a reduction in the relative error of these (non-utilization-dependent) functions. This observation continues to hold for the next two plots where we compare relative error by varying workload size (Figure 5.3) and resource period (Figure 5.4). We observe that for Suff, the relative error ranges from $25-95\%$ and it increases with the increase of workload size and resource period. For MC-A, relative error for both these cases are below 5%, and it is independent of workload size and resource period. In Figure 5.5, we compare the relative error of the two approximation algorithms (MC-A and BS-A-h) by varying the approximation parameter k. For both the cases, we observe that the relative error is very low (below 1%) even for moderate value of k (≥ 5). The relative error for BS-A-h is slightly higher than MC-A in all the above cases due to the fact that in the former case we have used a threshold of 10^{-6} while performing binary search of minimum capacity Θ. Further, in BS-A-h, the approximation of the special tasks which represent the resource unavailability period results a slight overestimation of minimum capacity by this algorithm.

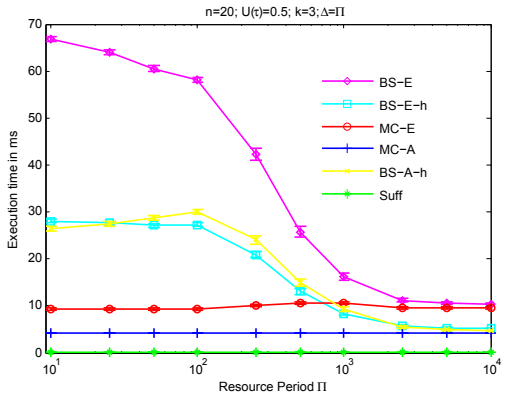


Figure 5.8: Execution Time vs Resource Period

Next, we compare the execution time (in ms) for all six algorithms varying system utilization, workload size and resource period. In Figure 5.6, execution time for these algorithms are plotted against system utilization. We observe that for the iterative convergence-based algorithms (BS-E, BS-E-h and BS-A-h), execution time decreases with increasing system utilization. This is due to the fact that the initial values of the response time for each task in task system τ (see Equation 5.4) is higher at the initial steps of the iterative convergence algorithm. This results in fewer number of iterations for the response time to converge, and thus reduces overall execution time of the algorithm. The execution time for the other three algorithms (i.e., MC-E, MC-A, Suff) does not vary much with respect to utilization. Notice that the comparison between the two heuristics based iterative algorithms: the approximate algorithm BS-A-h performs worse than the exact algorithm BS-E-h. This is due to the fact that the approximate response time obtained by BS-A-h is higher than that of the exact response time obtained by BS-E-h which results the former algorithm taking more iterations to converge. This is not trivial since at each iteration, the response time calculation for the approximate algorithm takes less time than the exact algorithm. Also, we observe similar run-time performance of BS-A-h in the last two plots.

In Figure 5.7, we compare execution time for all algorithms varying workload size, and observe that as the number of tasks in the system increases, the execution time for all the algorithms except the constant-time sufficient algorithm increases. However, the execution time for MC-E grows at higher rate than BS-E-h, and crosses it at around $n = 48$. This is due to the fact that MC-E calculates minimum capacity for each point in the testing set, and the size of testing set grows pseudo-polynomially with the workload size (Equation 3.7), whereas the growth of the iterative algorithms (BS-E, BS-E-h and BS-A-h) is proportional

to the workload size. Thus, we may conclude that the heuristic-based exact algorithm (BS-E-h) is more suitable when workload size is high. Finally, in Figure 5.8 execution time is compared against resource period. Here we observe that MC-E, MC-A and Suff do not vary much with resource period as expected. However, execution time for iterative exact algorithms decrease with increasing period, due to large initial response time for larger value of Π.

Although for moderate task system size the execution time of MC-E is very competitive to MC-A (Figure 5.6), while determining the interface parameters using the capacity determination algorithm as a subroutine of the period selection algorithm [44], it significantly adds up to the time required to determine interface parameters. Therefore, the near-optimal capacity determination algorithm MC-A can be used with very low relative error ($< 1\%$ for $k = 5$). In this chapter, we have considered constrain-deadline sporadic tasks in which case the execution time of the exact algorithm (MC-E) is proportional to $d_{\max}$ (i.e., maximum relative deadline among tasks). However, when task deadlines are arbitrary, this condition no longer holds and an approximate algorithm performs much better than the exponential time exact algorithm as shown in the next section.

5.4 Efficient Capacity Determination for Arbitrary Deadline FP-Scheduled Components

For arbitrary deadline sporadic tasks (i.e., task deadlines can be greater task periods) as components scheduled upon EDP resource, we characterize an exact schedulability condition (Theorem 9) for fixed-priority-scheduled components which potentially requires exponential runtime. To address the computational inefficiency, we are give a sufficient schedulability algorithm in this section.

When deadlines can exceed periods, it is no longer sufficient to check the response-times of only the first job ($\tau_{i,1}$) of each task [58]. Instead, Lehoczky [58] showed that is potentially necessary to check the response-time of all the jobs in *synchronous level-i busy interval* $\mathcal{B}_i$ for each task τ_i, which is the longest possible busy interval.

The *cumulative request-bound function* for job $\tau_{i,j}$ is defined as follows:

$$\mathsf{RBF}_{i,j}(t) \stackrel{\text{def}}{=} je_i + \sum_{h=1}^{i-1} \mathsf{rbf}(\tau_h, t). \tag{5.53}$$

The exact schedulability condition for arbitrary deadline fixed-priority scheduling in preemptive uniprocessor platform is given by the following equation.

$$\forall i, j \in \mathbb{N} : a_{ij} \in [0, \mathcal{B}_i) :: \exists t \in (a_{ij}, \bar{d}_{ij}], \mathsf{RBF}_{i,j}(\tau_i, t) \leq t \tag{5.54}$$

This condition needs to be verified at the following testing set[6] points (*ordered*):

$$\mathsf{TS}_i(\tau) \stackrel{\text{def}}{=} \left\{ t = b \cdot p_a : a = 1, \ldots, i; b = 1, \ldots, \left\lfloor \frac{\mathcal{B}_i}{p_a} \right\rfloor \right\}. \tag{5.55}$$

The size of this set may be as large as $\sum_{j=1}^{i} \left\lfloor \frac{\mathcal{B}_i}{p_j} \right\rfloor$ which is dependent on the task periods, and thus requires pseudo-polynomial time feasibility test. Again, at any instant t since d_i can be greater p_i for task τ_i, the number of active jobs of τ_i could be $\Theta(d_i/p_i)$, which is also pseudo-polynomial.

We can obtain an approximate cumulative request bound function using Equation 5.15 and 5.53 as follows:

$$\widetilde{\mathsf{RBF}}_{i,j}(t) \stackrel{\text{def}}{=} je_i + \sum_{h=1}^{i-1} \widetilde{\mathsf{rbf}}(\tau_h, t, k). \tag{5.56}$$

Fisher et al. [45] proposed an approximation to the feasibility test by approximating testing set points. A sporadic or synchronous periodic task system with arbitrary relative deadlines is schedulable if

$$\forall i, j \in \mathbb{N} : a_{ij} \in [0, \mathcal{B}_i) :: \exists t \in (a_{ij}, \bar{d}_{ij}], \widetilde{\mathsf{RBF}}_{i,j}(t) \leq t. \tag{5.57}$$

The testing set for this condition reduces to:

$$\widehat{\mathsf{TS}}_i(\tau, k) \stackrel{\text{def}}{=} \{ t = b \cdot p_a : a = 1, \ldots, i-1; b = 1, \ldots, k-1 \}. \tag{5.58}$$

Let t_a, t_{a+1} denote any pair of consecutive values in the above ordered set. $\widehat{\mathsf{TS}}_i(\tau, k)$ contains the points $t_1, t_2, \ldots, t_{(i-1)(k-1)}$. For convenience, we assume that the first point in the testing set, t_0 equals zero and the last point in the testing $t_{(i-1)(k-1)+1}$ equals ∞.

In the EDP resource model, the processor is no longer dedicated to a single component, but shared among numerous components. The length of a level-i busy interval for a task system τ now depends upon the amount of processing time that a resource Ω can provide to τ over that interval. Thus, we cannot use

[6]Note that we slightly abuse notation in this section to represent testing set similar to Section 5.3

Equation 5.54 or 5.1 to directly evaluate fixed-priority-schedulability upon Ω. Instead, we need to redefine the busy interval to account for the effects of Ω's resource parameters. Below we define the level-i busy interval with respect to EDP resource Ω.

Definition 14 (Synchronous Level-i Busy Interval for Ω) *A level-i busy interval upon an* EDP *resource $\Omega = (\Pi, \Theta, \Delta)$ is a time interval $[0, \mathcal{B}_i^\Omega)$ where only jobs of $\{\tau_1, \tau_2, \ldots, \tau_i\}$ are execute upon Ω and the following conditions hold:*

1. *All tasks of $\{\tau_1, \tau_2, \ldots, \tau_i\}$ release jobs at time zero.*
2. *Each task of $\{\tau_1, \tau_2, \ldots, \tau_i\}$ releases tasks periodically (i.e., $\tau_k \in \{\tau_1, \tau_2, \ldots, \tau_i\}$ has successive job releases p_k time units apart).*
3. The execution provided by Ω over any subinterval $[0, t) \subseteq [0, \mathcal{B}_i^\Omega)$ is minimal (i.e., the execution received by the component at $t \in [0, \mathcal{B}_i^\Omega)$ equals $\mathsf{sbf}(\Omega, t)$).
4. *$\mathcal{B}_i^\Omega$ is the first time instant such that all jobs of τ_i released in the interval $[0, \mathcal{B}_i^\Omega)$ have completed.*

Using the above definition, we may easily obtain the following theorem (using identical arguments as Lechoczky [56, 58]) which states an exact schedulability condition for EDP resource Ω where the arbitrary-deadline task system is scheduled by an fixed-priority-scheduling algorithm. Informally, it states that all jobs of task τ_i in the busy interval $\mathcal{B}_i^\Omega$ must complete at time t before its deadline.

Theorem 9 *An arbitrary-deadline sporadic task τ_i is fixed-priority-schedulable upon an* EDP *resource $\Omega = (\Pi, \Theta, \Delta)$, if and only if,*

$$\left(\forall j \in \mathbb{N}^+ : (j-1)p_i \leq \mathcal{B}_i^\Omega :: \exists t \in (a_{ij}, \bar{d}_{ij}], \mathsf{RBF}_{i,j}(\tau_i, t) \leq \mathsf{sbf}(\Omega, t)\right) \bigwedge \left(U_\tau \leq \frac{\Theta}{\Pi}\right). \tag{5.59}$$

While the above theorem gives a test for fixed-priority schedulability upon an EDP resource, the complexity of the test depends on the value of $\mathcal{B}_i^\Omega$ which can be quite large. Furthermore, the above theorem essentially requires simulation of the synchronous arrival sequence for τ_i up to $\mathcal{B}_i^\Omega$. For task system schedulability, we must repeat this process for each $\tau_i \in \tau$. We will address the lack of efficient schedulability results for this large and important class of task systems in our future work.

5.4.1 Schedulability Algorithm for FP-Scheduled Arbitrary-Deadline Tasks

In this section, we give our schedulability algorithm for arbitrary-deadline sporadic tasks on an EDP resource called AFPSCHEDULABILITY with a formal proof correctness.

We now define notation to represent the discontinuous line segments of the cumulative request bound function ($\widetilde{\mathsf{RBF}}_{i,j}$) (similar to section 5.3). Consider any $t \in \widehat{\mathsf{TS}}_i(\tau, k)$; define D_t to be request bound function at time t, that is the $\widetilde{\mathsf{RBF}}_{i,j}(t)$ of $\tau_{i,j}$. Define $\mathcal{L}^i_{t_a}$ to be the (non-vertical) line segment of the cumulative request bound function $\widetilde{\mathsf{RBF}}_{i,j}$ between consecutive testing set points t_a and t_{a+1}. Note that $\widetilde{\mathsf{RBF}}_{i,j}$ has a slope equal to the total utilization of all task τ_h with higher priority than τ_i i.e., $(h < i)$ and $t_{a+1} \geq (k-1)p_h$ (i.e., the total utilization of all tasks that are in the "approximate" portion of the $\widetilde{\mathsf{rbf}}$ after time point t_a). More formally,

$$\alpha^{i,j}_{t_a} \stackrel{\text{def}}{=} \sum_{\tau_h \in \tau : (t_{a+1} \geq (k-1)p_h) \wedge (h<i)} u_h. \tag{5.60}$$

The line segment (Figure 5.9) has a closed right endpoint $(t_{a+1}, D_{t_{a+1}})$ and an open left endpoint $(t_a, \bar{D}_{t_a})$ where

$$\bar{D}_{t_a} \stackrel{\text{def}}{=} \begin{cases} D_{t_{a+1}} + \alpha^{i,j}_{t_a}(t_a - t_{a+1}) & \text{if } t_{a+1} \geq (k-1)p_i, \\ je_i + \sum_{h=1}^{i-1} e_h + t_a \alpha^{i,j}_{t_a} & \text{otherwise.} \end{cases} \tag{5.61}$$

Thus, we formally define the line segment $\mathcal{L}^i_{t_a}$ as follows (similar to Equation 5.18).

$$\mathcal{L}^i_{t_a} \stackrel{\text{def}}{=} \{(x, y) \in \mathbb{R}^2 \mid (x \in (t_a, t_{a+1}]) \wedge (y = \bar{D}_{t_a} + \alpha^{i,j}_{t_a}(x - t_a))\}. \tag{5.62}$$

From the above definitions, it is straightforward to verify that for all $t \in (t_a, t_{a+1}]$, $\widetilde{\mathsf{RBF}}_{i,j}(t)$ is equivalent to $\mathcal{L}^i_{t_a}$. Furthermore, the following lemma is apparent from the observation that $\widetilde{\mathsf{RBF}}_{i,j}(t) \geq \mathsf{RBF}(\tau_i, t)$ for all $t > 0$.

Lemma 27 *For any $t \in \widehat{\mathsf{TS}}_i(\tau, k)$, $\widetilde{\mathsf{RBF}}_{i,j}(t) \leq D_t$ for all $(t, D_t) \in \mathcal{L}^i_{t_a}$.*

Definition 15 (Intersection of $\mathcal{L}^i_{t_a}$ and usbf) *For any line segment $\mathcal{L}^i_{t_a}$ of $\widetilde{\mathsf{RBF}}_{i,j}(t)$, the intersection of line $\mathcal{L}$, containing $\mathcal{L}^i_{t_a}$, with* usbf *is given by the following expression:*

$$\phi^{i,j}_{t_a} \stackrel{\text{def}}{=} \frac{D_{t_{a+1}} - \alpha^{i,j}_{t_a} t_{a+1} + \frac{\Theta}{\Pi}(\Delta - \Theta)}{\frac{\Theta}{\Pi} - \alpha^{i,j}_{t_a}}. \tag{5.63}$$

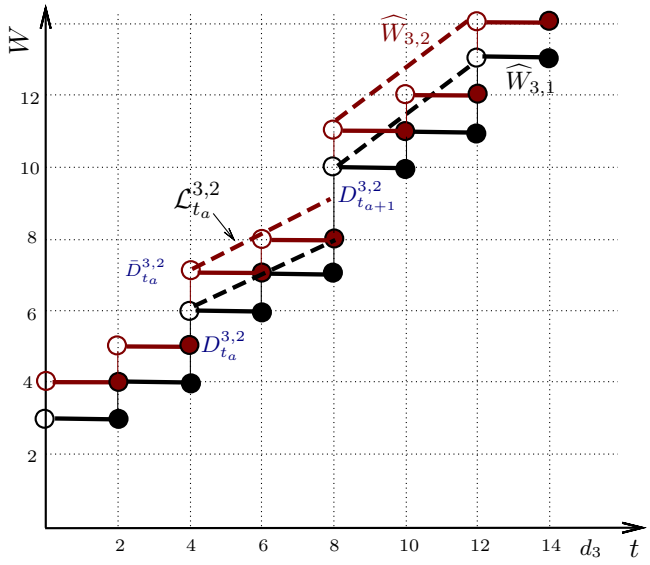


Figure 5.9: Consider τ consisting $\tau_1(1,5,2), \tau_2(1,10,4)$ and $\tau_3(1,15,8)$, where τ_1 and τ_2 have higher priority than τ_3. The cumulative request bound functions $W_{3,1}$ and $W_{3,2}$ are shown in the plot. The dashed lines represent approximation $\widetilde{\mathsf{RBF}}_{i,j}$ for each of the jobs $\tau_{i,j}$. A line segment $\mathcal{L}^i_{t_a}$ is any non-vertical line of $\widetilde{\mathsf{RBF}}_{i,j}$.

Note that by Equation 5.61, $\phi^{i,j}_{t_a}$ can also be calculated using $\bar{D}_{t_a}$ and $\alpha^{i,j}_{t_a}$. We now characterize the elapsed time between successive intersections of $\mathcal{L}^i_{t_a}$ and $\mathcal{L}^{i,j+1}_{t_a}$ with usbf. As the $\mathcal{L}^i_{t_a}$ and $\mathcal{L}^{i,j+1}_{t_a}$ correspond to the cumulative request bound function $\widetilde{\mathsf{RBF}}_{i,j}$ and $\widetilde{\mathsf{RBF}}_{i,j+1}$ over the interval $(t_a, t_{a+1}]$, we know (by Equation 5.56) that $\widetilde{\mathsf{RBF}}_{i,j+1}(t) - \widetilde{\mathsf{RBF}}_{i,j}(t) = e_i$ for all $t \in (t_a, t_{a+1}]$. Thus, the (horizontal) distance between the intersections of the two successive line segments with usbf can be found below.

$$\varphi^i_{t_a} \stackrel{\text{def}}{=} \phi^{i,j+1}_{t_a} - \phi^{i,j}_{t_a} = \frac{e_i}{\frac{\Theta}{\Pi} - \alpha^{i,j}_{t_a}} \tag{5.64}$$

The following lemma relates the concept of feasibility regions with the intersection of $\widetilde{\mathsf{RBF}}_{i,j}$ and sbf.

Lemma 28 *For any $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$ and $(t, D_t) \in \mathcal{L}^i_{t_a}$, the inequality $\widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{sbf}(\Omega, t)$ holds, if and only if there exists $\ell \in \mathbb{N}$ such that $(t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta, \Delta)$.*

The following two specially-defined steps (with respect to $\mathcal{L}^i_{t_a}$ and Ω) will be useful in constructing a

schedulability test:

$$\underline{\ell}_{t_a}^{i,j} \stackrel{\text{def}}{=} \max\left\{\left\lceil \frac{\bar{D}_{t_a}}{\Theta} \right\rceil, \left\lceil \frac{\bar{D}_{t_a} - \alpha_{t_a}^{i,j}(t_a - \Delta + \Theta)}{\Theta - \alpha_{t_a}^{i,j}\Pi} \right\rceil\right\} \tag{5.65}$$

and

$$\overline{\ell}_{t_a}^{i,j} \stackrel{\text{def}}{=} \left\lfloor \frac{t_{a+1} - D_{t_{a+1}} - \Delta + \Theta}{\Pi - \Theta} \right\rfloor. \tag{5.66}$$

Algorithm Description

The pseudocode of our sufficient algorithm AFPSCHEDULABILITY is presented in Algorithm 4, along with an auxiliary functions AFPSINTERSTAGE, given in Algorithms 5. AFPSCHEDULABILITY works as follows. For each task $\tau_i \in \tau$, we try to determine which jobs $\tau_{i,j}$ in the synchronous level-i busy interval for Ω can complete by their deadline, starting with $j = 1$. We check job schedulability by considering each of the testing set points $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$ (in increasing order) and then incrementing j once we have determined that the previous job (or jobs) are schedulable. For each testing point and current job $\tau_{i,j}$ for which we are trying to determine schedulability, we consider the corresponding line segment $\mathcal{L}_{t_a}^i$. Line 5 informally checks whether an intersection between $\mathcal{L}_{t_a}^i$ and sbf is in the interval $(t_a, t_{a+1}]$. If not, we check (Line 6) whether $\tau_{i,j}$'s deadline elapses in the interval $(t_a, t_{a+1}]$, in which case we declare that the system is not schedulable.

In the case that $\mathcal{L}_{t_a}^i$ intersects with sbf over $(t_a, t_{a+1}]$, we determine the intersection point of $\mathcal{L}_{t_a}^i$ with the $\underline{\ell}_{t_a}^{i,j}$-step of the sbf in Line 10. Line 12 checks whether this intersection occurs prior to $\tau_{i,j}$'s deadline. If so, we then calculate in Line 15 the number of jobs after $\tau_{i,j}$ that complete execution (not necessarily by their deadline) in the interval $(t_a, t_{a+1}]$. Line 16 determines (via AFPSINTERSTAGE) whether these jobs will meet their deadline and returns *not schedulable*, if they do not. If each job that completes meets its deadline, we increment the variable j by the number of completed active jobs in Line 17, to update which jobs have finished by their deadline. If $j > \lceil \frac{t_{a+1}}{p_i} \rceil$, we can conclude that the end of the level-i busy interval occurs prior to t_{a+1} and that τ_i is schedulable and continue the test for the next task. Otherwise, we repeat the above process iterating over $\widehat{\mathsf{TS}}_i(\tau, k)$ until we check all jobs up until the final value of $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$. Line 27 checks (via AFPSINTERSTAGE) whether all jobs released after the final testing set point complete by their deadline. If so, we can conclude τ_i is schedulable and move onto the next task in τ.

Algorithm 4 AFPSCHEDULABILITY(Ω, τ, k)

Require: EDP Resource $\Omega = (\Pi, \Theta, \Delta)$; Task System τ; k.
Ensure: τ is FP-schedulable upon Ω.

1: **for all** $\tau_i \in \tau$ **do**
2: $j \leftarrow 1$
3: {*Consider testing set points in increasing order*}
4: **for all** $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$ **do**
5: **if** $\underline{\ell}_{t_a}^{i,j} > \overline{\ell}_{t_a}^{i,j}$ **then**
6: **if** $t_{a+1} \geq \bar{d}_{ij}$ **then**
7: **Return** *Not Schedulable*.
8: **end if**
9: **else**
10: $\hat{t} \leftarrow \frac{D_{t_{a+1}} - \alpha_{t_a}^{i,j} t_{a+1} + \underline{\ell}_{t_a}^{i,j} \Pi + \Delta - (\underline{\ell}_{t_a}^{i,j} + 1)\Theta}{1 - \alpha_{t_a}^{i,j}}$
11: **if** $\hat{t} \leq t_{a+1}$ **then**
12: **if** $\hat{t} > \bar{d}_{ij}$ **then**
13: **Return** *Not Schedulable*
14: **else**
15: $z \leftarrow \min\left\{\left\lceil \frac{t_{a+1}}{p_i} \right\rceil - j, \left(\left\lfloor \frac{\mathsf{sbf}(\Omega, t_{a+1}) - D_{t_{a+1}}}{e_i} \right\rfloor\right)_0\right\}$
16: AFPSINTERSTAGE$(\Omega, \tau_i, \tau_{i,j+1}, t_{a+1})$
17: $j \leftarrow j + z + 1$
18: **if** $j > \lceil \frac{t_{a+1}}{p_i} \rceil$ **then**
19: {*No active jobs, end of busy period*}
20: **Continue** Outer Loop.
21: **end if**
22: **end if**
23: **end if**
24: **end if**
25: **end for**{*End Inner Loop*}
26: {*Check final line segment.*}
27: AFPSINTERSTAGE$(\Omega, \tau_i, \tau_{i,j}, t_{(i-1)(k-1)+1})$
28: **end for**{*End Outer Loop*}
29: **Return** *Schedulable*

Algorithm Complexity

Each iteration of the "inner" loop of AFPSCHEDULABILITY from Lines 4 to 25 requires constant time (including the auxiliary function). The number times this inner loop executes for a given τ_i is equal to $|\widehat{\mathsf{TS}}_i(\tau, k)| = (i-1)(k-1)$. Furthermore, if $O(ik \cdot \lg ik)$ is required to sort each $\widehat{\mathsf{TS}}_i(\tau, k)$, the total complexity of the algorithm is given by $O(kn^2 \lg nk)$.

Algorithm 5 AFPSINTERSTAGE$(\Omega, \tau_i, \tau_{i,j}, t_a)$

Require: EDP Resource Ω; Task τ_i; Job $\tau_{i,j}$; Time t_a.
Ensure: All jobs $\tau_{i,j}$ and beyond of task τ_i are FP-schedulable in the interval $(t_a, t_{a+1}]$ upon Ω.
1: $\{\phi_{t_a}^{i,j}$ *and* $\varphi_{t_a}^{i}$ *are defined in Equations 5.63 and 5.64.*$\}$
2: **if** $\bar{d}_{ij} - \phi_{t_a}^{i,j} < \frac{\Pi - \Theta}{1 - \alpha_{t_a}^{i,j}}$ **then**
3: **Return** *Not Schedulable*
4: **end if**
5: **if** $p_i < \varphi_{t_a}^{i}$ **then**
6: **Return** *Not Schedulable*
7: **end if**

Algorithm Correctness

The main correctness result for our sufficient schedulability algorithm is given in the theorem below.

Theorem 10 *A sporadic task system τ is* FP*-schedulable upon an* EDP *resource Ω, if* AFPSCHEDULABILITY(Ω, τ, k) *returns "Schedulable".*

To prove the above theorem, we need to prove some additional lemmas. We start with a lemma that states schedulability condition for a line segment $\mathcal{L}_{t_a}^{i}$ of $\widetilde{\mathsf{RBF}}_{i,j}$ of $\tau_{i,j}$ with EDP resource Ω. In the previous section, we derived bounds for the value of Θ such that a line segment corresponding to τ_i's first of τ_{i1} is schedulable; i.e., if the necessary and sufficient conditions are satisfied, then $\widehat{W}_{i1}(t) \leq \mathsf{sbf}(\Omega, t)$ for some $t \in (t_a, t_{a+1}]$. We restate Lemma 14 to obtain bounds on the values of ℓ.

Lemma 29 *For any $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$ line segment $\mathcal{L}_{t_a}^{i}, \exists t \in (t_a, t_{a+1}] : \widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{sbf}(\Omega, t)$, if and only if there exists $\ell \in \mathbb{N}^+$ such that the following conditions satisfy.*

$$\begin{aligned} \ell &\leq \tfrac{t_{a+1} - D_{t_{a+1}} - \Delta + \Theta}{\Pi - \Theta}, && (5.67a) \\ \ell &\geq \tfrac{\bar{D}_{t_a}}{\Theta}, && (5.67b) \\ \ell &\geq \tfrac{\bar{D}_{t_a} - \alpha_{t_a}^{i,j}(t_a - \Delta + \Theta)}{\Theta - \alpha_{t_a}^{i,j}\Pi} && (5.67c). \end{aligned} \tag{5.67}$$

The next corollary follows immediately from the above lemma and Equations 5.65 and 5.66. The corollary shows that we can check whether $\mathcal{L}_{t_a}^{i}$ intersects with the sbf over $(t_a, t_{a+1}]$ by evaluating whether the condition $\underline{\ell}_{t_a}^{i,j} > \overline{\ell}_{t_a}^{i,j}$ is false (Line 5 of AFPSCHEDULABILITY).

Corollary 13 *For any $t_a \in \widehat{\mathsf{TS}}_i(\tau, k)$ and line segment $\mathcal{L}^i_{t_a}$ and $(t, D_t) \in \mathcal{L}^i_{t_a}$, $\exists t : \widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{sbf}(\Omega, t)$ if and only if $\underline{\ell}^{i,j}_{t_a} \leq \overline{\ell}^{i,j}_{t_a}$.*

The next lemma shows that if any part of $\mathcal{L}^i_{t_a}$ intersects with the sbf the point calculated in Line 10 of AFPSCHEDULABILITY also does.

Lemma 30 *For any $t_a \in \widehat{\mathsf{TS}}_i(\tau, k), \ell \in \mathbb{N}^+$, $\mathcal{L}^i_{t_a}$, Π, Θ and Δ, if $\underline{\ell}^{i,j}_{t_a} \leq \ell \leq \overline{\ell}^{i,j}_{t_a}$ and $\exists (t, D_t) : (t, D_t) \in \mathcal{L}^i_{t_a} \wedge (t, D_t) \in \mathcal{F}_\ell(\Pi, \Theta, \Delta)$ then $\exists (\hat{t}, D_{\hat{t}}) : (\hat{t}, D_{\hat{t}}) \in \mathcal{L}^i_{t_a} \wedge (\hat{t}, D_{\hat{t}}) \in \mathcal{F}_{\underline{\ell}^{i,j}_{t_a}}(\Pi, \Theta, \Delta)$ where*

$$\hat{t} \stackrel{def}{=} \frac{D_{t_{a+1}} - \alpha^{i,j}_{t_a} t_{a+1} + \underline{\ell}^{i,j}_{t_a} \Pi + \Delta - (\underline{\ell}^{i,j}_{t_a} + 1)\Theta}{1 - \alpha^{i,j}_{t_a}}.$$

Proof: We need to show for the point $(\hat{t}, D_{\hat{t}})$, it is in $\mathcal{F}_{\underline{\ell}^{i,j}_{t_a}}$-feasibility region. By definition of feasibility region (Definition 11), we have to show the following.

$$\begin{aligned} & \Theta \geq \tfrac{D_{\hat{t}} - \hat{t} + \underline{\ell}^{i,j}_{t_a} \Pi + \Delta}{\underline{\ell}^{i,j}_{t_a} + 1} && (5.68a) \\ \wedge \quad & \Theta \geq \tfrac{D_{\hat{t}}}{\underline{\ell}^{i,j}_{t_a}} && (5.68b) \end{aligned} \tag{5.68}$$

We start with the right hand side of 5.68a.

$$\begin{aligned} & \tfrac{D_{\hat{t}} - \hat{t} + \underline{\ell}^{i,j}_{t_a} \Pi + \Delta}{\underline{\ell}^{i,j}_{t_a} + 1} \\ = & \tfrac{\bar{D}_{t_a} + \alpha^{i,j}_{t_a}(\hat{t} - t_a) - \hat{t} + \underline{\ell}^{i,j}_{t_a} \Pi + \Delta}{\underline{\ell}^{i,j}_{t_a} + 1} \\ = & \tfrac{\bar{D}_{t_a} - \alpha^{i,j}_{t_a} t_a}{\underline{\ell}^{i,j}_{t_a} + 1} \\ & - \tfrac{(1 - \alpha^{i,j}_{t_a}) \frac{D_{t_{a+1}} - \alpha^{i,j}_{t_a} t_{a+1} + \underline{\ell}^{i,j}_{t_a} \Pi + \Delta - (\underline{\ell}^{i,j}_{t_a} + 1)\Theta}{1 - \alpha^{i,j}_{t_a}} - \underline{\ell}^{i,j}_{t_a} \Pi - \Delta}{\underline{\ell}^{i,j}_{t_a} + 1} \\ = & \tfrac{D_{t_{a+1}} - \alpha^{i,j}_{t_a} t_{a+1} - D_{t_{a+1}} + \alpha^{i,j}_{t_a} t_{a+1} - \underline{\ell}^{i,j}_{t_a} \Pi - \Delta}{\underline{\ell}^{i,j}_{t_a} + 1} + \tfrac{(\underline{\ell}^{i,j}_{t_a} + 1)\Theta + \underline{\ell}^{i,j}_{t_a} \Pi + \Delta}{\underline{\ell}^{i,j}_{t_a} + 1} \\ \leq & \Theta \end{aligned}$$

Thus, the first condition of feasibility region is verified.

By 5.67c, we know

$$\begin{aligned} & \underline{\ell}_{t_a}^{i,j} \geq \frac{\bar{D}_{t_a} - \alpha_{t_a}^{i,j}(t_a - \Delta + \Theta)}{\Theta - \alpha_{t_a}^{i,j}\Pi} \\ \Rightarrow \quad & \underline{\ell}_{t_a}^{i,j}(\Theta - \alpha_{t_a}^{i,j}\Pi) \geq D_{t_{a+1}} - \alpha_{t_a}^{i,j}(t_{a+1} - \Delta + \Theta) \end{aligned}$$

Next, we need to prove 5.68b.

$$\begin{aligned} \frac{D_{\hat{t}}}{\underline{\ell}_{t_a}^{i,j}} &= \frac{\bar{D}_{t_a} + \alpha_{t_a}^{i,j}(\hat{t} - t_a)}{\underline{\ell}_{t_a}^{i,j}} \\ &= \frac{D_{t_{a+1}} - \alpha_{t_a}^{i,j} t_{a+1} + \alpha_{t_a}^{i,j} \frac{D_{t_{a+1}} - \alpha_{t_a}^{i,j} t_{a+1} + \underline{\ell}_{t_a}^{i,j}\Pi + \Delta - (\underline{\ell}_{t_a}^{i,j}+1)\Theta}{1 - \alpha_{t_a}^{i,j}}}{\underline{\ell}_{t_a}^{i,j}} \\ &= \frac{D_{t_{a+1}} - \alpha_{t_a}^{i,j} t_{a+1} + \alpha_{t_a}^{i,j}(\underline{\ell}_{t_a}^{i,j}\Pi + \Delta - (\underline{\ell}_{t_a}^{i,j}+1)\Theta)}{(1 - \alpha_{t_a}^{i,j})\underline{\ell}_{t_a}^{i,j}} \\ &= \frac{D_{t_{a+1}} - \alpha_{t_a}^{i,j}(t_{a+1} + \Theta - \Delta) + \alpha_{t_a}^{i,j}\underline{\ell}_{t_a}^{i,j}\Pi - \alpha_{t_a}^{i,j}\underline{\ell}_{t_a}^{i,j}\Theta}{(1 - \alpha_{t_a}^{i,j})\underline{\ell}_{t_a}^{i,j}} \\ &\leq \frac{\underline{\ell}_{t_a}^{i,j}(\Theta - \alpha_{t_a}^{i,j}\Pi) + \alpha_{t_a}^{i,j}\underline{\ell}_{t_a}^{i,j}\Pi - \alpha_{t_a}^{i,j}\underline{\ell}_{t_a}^{i,j}\Theta}{(1 - \alpha_{t_a}^{i,j})\underline{\ell}_{t_a}^{i,j}} \\ &\leq \Theta \end{aligned}$$

The last line follows by using the fact derived from 5.67c. Thus, the point $(\hat{t}, D_{\hat{t}})$ is in $\mathcal{F}_{\underline{\ell}_{t_a}^{i,j}}$-feasibility region. ■

Lemma 31 *If* $\exists (t, D_t) \in \mathcal{L}_{t_a}^i \wedge t \in (a_{ij}, \bar{d}_{ij}] : (t, D_t) \in \mathcal{F}_{\underline{\ell}_{t_a}^{i,j}}(\Pi, \Theta, \Delta)$ *then* $\tau_{i,j}$ *is schedulable.*

Proof: By Lemma 28, if $(t, D_t) \in \mathcal{L}_{t_a}^i$ and $(t, D_t) \in \mathcal{F}_{\underline{\ell}_{t_a}^{i,j}}(\Pi, \Theta, \Delta)$, then approximate cumulative request for $\tau_{i,j}$ will be less or equal the supply at time t. By Theorem 9, task τ_i is schedulable if and only if for all job $\tau_{i,j}$ in the busy interval $\mathcal{B}_i^{\Omega}$, the cumulative request $\widetilde{\mathsf{RBF}}_{i,j}(t)$ is less or equal supply $\mathsf{sbf}(\Omega, t)$ for some point $t \in (a_{ij}, \bar{d}_{ij}]$. Therefore, combining the two statements above, a job $\tau_{i,j}$ will be schedulable if and only if $\exists (t, D_t) \in \mathcal{L}_{t_a}^i$ such that $(t, D_t) \in \mathcal{F}_{\underline{\ell}_{t_a}^{i,j}}(\Pi, \Theta, \Delta)$. ■

The usbf and the steps of sbf form obtuse triangular regions with base as the plateau of an sbf step, and the other two sides are as usbf (longest side) and the slope of sbf (see Figure 5.10). As $\mathsf{usbf} \geq \mathsf{sbf}$, a line with slope ≥ 0 entering and leaving the triangular region must have first intersection with the usbf, and then it will exit the region with an intersection with the sbf slope portion of same triangular region.

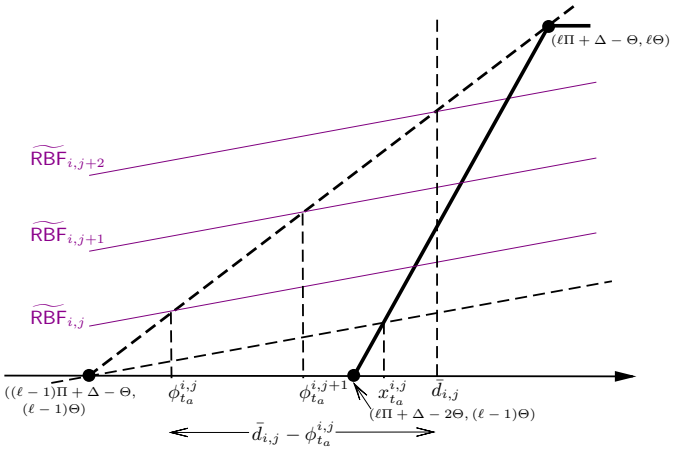


Figure 5.10: Illustration of AFPSINTERSTAGE and triangular region between usbf and sbf.

We denote the line containing the line segment $\mathcal{L}^i_{t_a}$ by $f(x) = \alpha^{i,j}_{t_a}(x - t_a) + \bar{D}_{t_a}$. Now consider the intersection of $f(x)$ and the triangular region formed by the usbf and some $\ell - 1$ and ℓ steps of the sbf. Let the horizontal distance between the intersection of $f(x)$ and usbf and the intersection of $f(x)$ and the sloped-portion of the ℓ step of the sbf be denoted by $x^{i,j}_{t_a}$. The next lemma obtains an upper bound on $x^{i,j}_{t_a}$.

Lemma 32 *For any line segment $\mathcal{L}^i_{t_a}$ of $\widetilde{\mathsf{RBF}}_{i,j}(t)$, longest horizontal distance between the points at which the line intersects with the* usbf(Ω, t) *and* sbf(Ω, t) *satisfies*

$$x^{i,j}_{t_a} \leq \frac{\Pi - \Theta}{1 - \alpha^{i,j}_{t_a}}. \tag{5.69}$$

Proof: Consider the set of all lines parallel to $\mathcal{L}^i_{t_a}$ that intersect with the triangular region formed from usbf(Ω, t) and $\ell - 1$ and ℓ steps of sbf(Ω, t) by first passing through the usbf face of the triangle. Let $g(x) \stackrel{\text{def}}{=} x \cdot \alpha^{i,j}_{t_a} + b$ for some $b \in \mathbb{R}$ be the line where the horizontal distance is maximal. The leftmost point for the triangle is $((\ell - 1)\Pi + \Delta - \Theta, (\ell - 1)\Theta)$ and the slope of sbf is given by the equation $h(x) \stackrel{\text{def}}{=} x - \ell\Pi - \Delta + (\ell + 1)\Theta$. By definition of g and h, we obtain the intersection $s \stackrel{\text{def}}{=} \frac{b + \ell\Pi + \Delta - (\ell+1)\Theta}{1 - \alpha^{i,j}_{t_a}}$. The intersection of $\mathcal{L}^i_{t_a}$ and usbf is $q \stackrel{\text{def}}{=} \frac{b + \frac{\Theta}{\Pi}(\Delta - \Theta)}{\frac{\Theta}{\Pi} - \alpha^{i,j}_{t_a}}$. Since g has the maximal distance, it must be that

$$\begin{aligned} x^{i,j}_{t_a} \leq s - q \quad &= \frac{b + \ell\Pi + \Delta - (\ell+1)\Theta}{1 - \alpha^{i,j}_{t_a}} - \frac{b + \frac{\Theta}{\Pi}(\Delta - \Theta)}{\frac{\Theta}{\Pi} - \alpha^{i,j}_{t_a}} \\ &= \frac{(\Pi - \Theta)(\ell\Theta - \alpha^{i,j}_{t_a}(\ell\Pi + \Delta - \Theta) - b)}{(1 - \alpha^{i,j}_{t_a})(\Theta - \alpha^{i,j}_{t_a}\Pi)} \end{aligned} \tag{5.70}$$

The only value above that is not fixed is b. Therefore, by finding the b that maximizes the right-hand side of Equation 5.70, we can obtain an upper bound on $x_{t_a}^{i,j}$. The first derivative of the right-hand-side expression with respect to b is $\frac{-(\Pi-\Theta)}{(1-\alpha_{t_a}^{i,j})(\Theta-\alpha_{t_a}^{i,j}\Pi)}$. As $\Pi \geq \Theta$, $\frac{\Theta}{\Pi} \geq \alpha_{t_a}^{i,j}$, and $0 \leq \alpha_{t_a}^{i,j} \leq 1$, the derivative is negative, thus, the largest value of $x_{t_a}^{i,j}(b)$ can be obtained for the smallest value of b. The line with smallest y-intercept b of any line parallel to $\mathcal{L}_{t_a}^i$ entering the triangle through the usbf face must pass through the point $((\ell-1)\Pi+\Delta-\Theta,(\ell-1)\Theta)$ of the triangle. Putting this point in the equation of $\mathcal{L}_{t_a}^i$, we obtain $b=(\ell-1)\Theta-\alpha_{t_a}^{i,j}((\ell-1)\Pi+\Delta-\Theta)$, and from Eqn. 5.70, we obtain upper bound on $x_{t_a}^{i,j}$. ■

The following lemma proves the correctness of the number of active jobs at any interval $(t_a, t_{a+1}]$ determined by the algorithm in Line 15.

Lemma 33 *For $t_a \in \widehat{\mathsf{TS}}_i(\tau,k)$ and $i,j \in \mathbb{N}^+$, if all jobs from 1 to $j-1$ meet their deadline, and $\tau_{i,j}$ is the first job to meet its deadline in the interval $(t_a,t_{a+1}]$, then the number of jobs that complete execution in the interval is at least* $\min\left\{\lceil\frac{t_{a+1}}{p_i}\rceil - j, \left(\left\lfloor\frac{\mathsf{sbf}(\Omega,t_{a+1})-\bar{D}_{t_a}}{e_i}\right\rfloor\right)_0\right\}$.

Proof: At any time point t_{a+1}, the total number of jobs released by task τ_i is given by $\lceil\frac{t_{a+1}}{p_i}\rceil$. If $\tau_{i,j}$ is the first job to met its deadline in the interval $(t_a,t_{a+1}]$, then it is the lowest active job at the beginning of the interval. Thus, the term $\lceil\frac{t_{a+1}}{p_i}\rceil - j$ denotes remaining active jobs in the interval.

We now determine the number of such jobs that complete execution. As $\tau_{i,j}$ met its deadline, let $\mathcal{L}_{t_a}^i$ be the line segment such that $\exists(t,D_t) \in \mathcal{L}_{t_a}^i, \widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{sbf}(\Omega,t)$. To check whether subsequent jobs of τ_i complete in $(t_a,t_{a+1}]$, we must check if subsequent parallel line segments $\mathcal{L}_{t_a}^{i,j+1}, \mathcal{L}_{t_a}^{i,j+2}, \ldots$ intersect with sbf prior to time t_{a+1}. Since t_{a+1} is the latest point of intersection with sbf that we consider, all line segments must be below the parallel line denoted $\mathcal{L}$ that intersects with the point $(t_{a+1}, \mathsf{sbf}(\Omega,t_{a+1}))$; see Figure 5.11. Furthermore, the line segments of that intersect with sbf prior to t_{a+1} must also intersect with usbf prior to $\mathcal{L}$'s intersection with usbf which may be derived as:

$$t_1 = \frac{\mathsf{sbf}(\Omega,t_{a+1}) - \alpha_{t_a}^{i,j}t_{a+1} + \frac{\Theta}{\Pi}(\Delta-\Theta)}{\frac{\Theta}{\Pi} - \alpha_{t_a}^{i,j}} \tag{5.71}$$

By Definition 15, $\phi_{t_a}^{i,j}$ is the intersection of $\mathcal{L}_{t_a}^i$ with usbf, and subsequent jobs' request bound function intersect usbf at $\varphi_{t_a}^i$ interval. We can obtain number of active jobs after $\tau_{i,j}$ that can intersect with the usbf in the interval $[\phi_{t_a}^{i,j}, t_1]$, which is $\left(\left\lfloor\frac{t_1-\phi_{t_a}^{i,j}}{\varphi_{t_a}^i}\right\rfloor\right)_0$ or $\left(\left\lfloor\frac{\mathsf{sbf}(\Omega,t_{a+1})-D_{t_{a+1}}}{e_i}\right\rfloor\right)_0$. This is the upper bound

on the number of jobs whose request can be satisfied within the interval. As this value cannot exceed the number of active jobs of τ_i available in the system at time t_{a+1}, we obtain actual number of jobs satisfying their request in the interval by taking $\min\left\{\lceil\frac{t_{a+1}}{p_i}\rceil - j, \left(\left\lfloor\frac{\mathsf{sbf}(\Omega,t_{a+1})-D_{t_{a+1}}}{e_i}\right\rfloor\right)_0\right\}$. ■

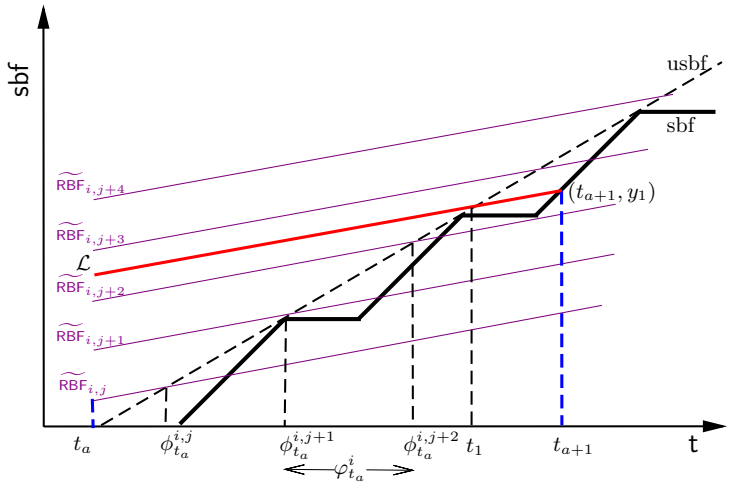


Figure 5.11: Determining number of active jobs in $(t_{a-1}, t_a]$ interval.

We need two additional lemmas to prove the correctness of AFPSINTERSTAGE$(\Omega, \tau_i, \tau_{i,j})$. The first lemma shows that if the condition of Line 2 is not satisfied, it is safe to conclude that $\tau_{i,j}$ does not miss its deadline.

Lemma 34 *For $\forall i, j$, if $\bar{d}_{ij} - \phi^{i,j}_{t_a} \geq \frac{\Pi-\Theta}{1-\alpha^{i,j}_{t_a}}$, then $\tau_{i,j}$ does not miss its deadline in the interval $(t_a, t_{a+1}]$.*

Proof: The job $\tau_{i,j}$ is schedulable if it meets its deadline; i,e., there exists a point $t \in [a_{ij}, \bar{d}_{ij}]$ such that $\widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{sbf}(\Omega, t)$. By Definition 15, for $\tau_{i,j}$, $\phi^{i,j}_{t_a}$ is a point at which $\widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{usbf}(\Omega, t)$.

We prove the proposition by taking contrapositive, that is, if $\tau_{i,j}$ is not schedulable then $\bar{d}_{ij} - \phi^{i,j}_{t_a} < \frac{\Pi-\Theta}{1-\alpha^{i,j}_{t_a}}$. By Lemma 32, $x^{i,j}_{t_a}$ is the horizontal distance after $\phi^{i,j}_{t_a}$ to obtain a point such that $\widetilde{\mathsf{RBF}}_{i,j}(t) \leq \mathsf{sbf}(\Omega, t)$ and $x^{i,j}_{t_a} \leq \frac{\Pi-\Theta}{1-\alpha^{i,j}_{t_a}}$. Thus, if the job missed deadline, than $\phi^{i,j}_{t_a} + x^{i,j}_{t_a} > \bar{d}_{ij} \Rightarrow \phi^{i,j}_{t_a} + \frac{\Pi-\Theta}{1-\alpha^{i,j}_{t_a}} > \phi^{i,j}_{t_a}$. Rearrangement proves the lemma. ■

The next lemma shows that if the condition of Line 5 of AFPSINTERSTAGE$(\Omega, \tau_i, \tau_{i,j})$ is not satisfied no active job of τ_i will miss its deadline in the interval $(t_a, t_{a+1}]$.

Lemma 35 *Let $\tau_{i,j}$ be an active job in the interval $(t_a, t_{a+1}]$ such that $\bar{d}_{ij} - \phi^{i,j}_{t_a} \geq \frac{\Pi-\Theta}{1-\alpha^{i,j}_{t_a}}$, then no active job in the interval after $\tau_{i,j}$ will miss its deadline if $p_i \geq \varphi^i_{t_a}$.*

Proof: Assume that $\tau_{i,j}, \tau_{i,j+1}, \ldots, \tau_{i,j+s}$ are the active jobs in the interval $(t_a, t_{a+1}]$. By Lemma 34, $\bar{d}_{ij} - \phi_{t_a}^{i,j} \geq \frac{\Pi - \Theta}{1 - \alpha_{t_a}^{i,j}}$ implies $\tau_{i,j}$ does not miss its deadline. For the next job $\tau_{i,j+1}$, the difference between its absolute deadline and intersection with usbf is $d_{i,j+1} - \phi_{i,j+1} = \bar{d}_{ij} + p_i + \phi_{t_a}^{i,j} + \varphi_{t_a}^{i} = \bar{d}_{ij} - \phi_{t_a}^{i,j} + (p_i - \varphi_{t_a}^{i})$. Similarly, for $\tau_{i,j+2}$, distance is $d_{i,j+2} - \phi_{t_a}^{i,j+2} = \bar{d}_{ij} - \phi_{t_a}^{i,j} + 2(p_i - \varphi_{t_a}^{i})$. Notice the distance between intersection of the corresponding line segment and usbf grows by $p_i - \varphi_{t_a}^{i}$. Thus, $p_i \geq \varphi_{t_a}^{i}$ implies that the growth is positive and the distance between deadline and intersection with usbf is increasing for active jobs later than $\tau_{i,j}$. Furthermore, by Lemma 34, the distance is of $d_{i,j+k} - \phi_{t_a}^{i,j+k}$ is at least $\frac{\Pi - \Theta}{1 - \alpha_{t_a}^{i,j}}$ for each $k = 1, \ldots, s$, which implies that each $\tau_{i,j+k}$ does not miss a deadline in the interval $(t_a, t_{a+1}]$. ■

We are ready to prove the correctness of our algorithm.

Proof of Theorem 10 To prove the correctness of AFPSCHEDULABILITY(Ω, τ, k), we need to prove the following loop invariant for the inner loop (Line 4- 25).

At the beginning of the inner loop, all the jobs from 1 *to* $j - 1$ *are schedulable.*

Initialization When $j = 1$, there is no job in the range 1 to $j - 1$. We can conclude that all job in that range is schedulable (trivially).

Maintenance At each loop iteration, we are starting with $\tau_{i,j}$, which is the lowest active job in the interval $(t_a, t_{a+1}]$ (has arrival at or before t_a and deadline after t_a). All the jobs from 1 to $j - 1$ are schedulable. If any of them missed deadline, the algorithm would not have continued to j-th job. Now we need to show after current iteration we have successfully checked schedulability of the jobs that are active within current interval $(t_a, t_{a+1}]$. We check the schedulability condition for the first active job ($\tau_{i,j}$) in the interval (Line 5 to 13). If $\tau_{i,j}$ does not miss its deadline in this interval, then we obtain minimum number of active jobs that must finish their execution in the interval (Line 15), and perform sufficient schedulability test for these jobs (Line 16). If these conditions are satisfied (Lemma 34, 35) then j is updated (Line 17) by the job which is active, that is, the execution request of the job is not satisfied within current interval $(t_a, t_{a+1}]$.

At the end of the loop we check if all the jobs of τ_i available in the system have meet their execution requirement, and terminate the loop (Line 18) for this case.

Termination The loop terminates when all intervals of testing set points have been considered, or the busy interval for task τ_i have ended. As the loop did not return *Not Schedulable*, all the jobs from 1 to

$j-1$ are schedulable.

The outer loop checks the schedulability condition for each task in the task system. ■

5.4.2 Simulation Results

We compared our proposed algorithm with the exact and the approximate schedulability tests for arbitrary-deadline FP-scheduled task system with EDP resource. From the compositional system scheduled upon EDP resource, we obtained an equivalent dedicated uniprocessor system for each component by adding a special highest priority task to the task system τ, which represents the no-supply period of the EDP resource (similar to [71]). Then we performed response-time based exact [58] and approximate [45] schedulability test for dedicated uniprocessor system similar to the approach of Section 5.2 to the modified task system. We have considered a fixed resource period Π and deadline Δ, and computed minimum capacity Θ required to schedule the system for the three tests. The simulation parameters and value ranges are shown below:

1. Task system size $n = 10$; k is set in the range $[1, 20]$; Π is set to 10 and Δ is equal to Π.
2. U_τ is taken from the range $[0.1, 0.9]$ at 0.05-increments and individual task utilizations u_i are generated using UUniFast algorithm [25].
3. Each $\tau_i \in \tau$, period p_i is uniformly drawn from the interval $[5, 30]$, deadline d_i is uniformly drawn from $[5, 100]$ and execution time e_i is set to $u_i.p_i$.
4. The component-level scheduler is rate-monotonic.
5. A 2.2GHz Intel Core i7-2670QM CPU with 8Gb RAM is used. Each point in the plot represents 1000 simulation runs.

For each simulation, given task system size n and system utilization U_τ, we randomly generated taskset parameters u_i, p_i, d_i, for each task τ_i. Then we executed a binary search to search for minimum capacity in the range $[0, \Pi]$, using each of the schedulability test as subroutines to generate exact, approximate and sufficient (i.e., AFPSFEASIBILITY) solutions.

In Figure 5.12 the capacity obtained from the three algorithms are plotted against system utilization. We observe that the sufficient test performs better than the approximate test. In Figure 5.13, the execution

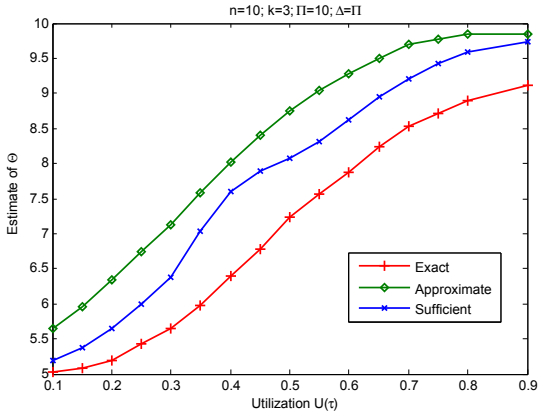


Figure 5.12: Resource Capacity vs Utilization

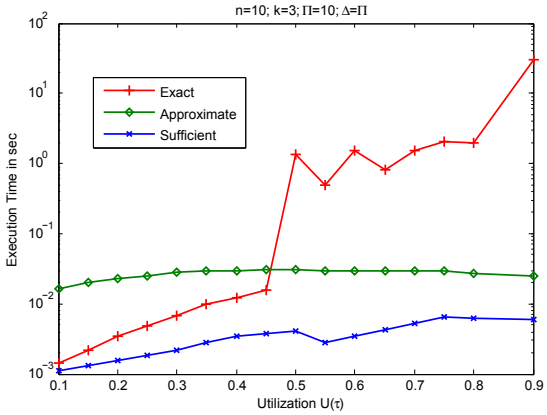


Figure 5.13: Execution Time vs Utilization

time (in log scale) for the three algorithms are plotted against task system utilization. Note that the capacity Θ is chosen considering the resource starvation period (i.e., $\Theta \geq \Pi/2$) and since task deadlines can exceed task periods, the iterative exact algorithm converges quickly for lower system utilization. However, for higher system utilization, the execution time of the sufficient test outperforms the other two tests.

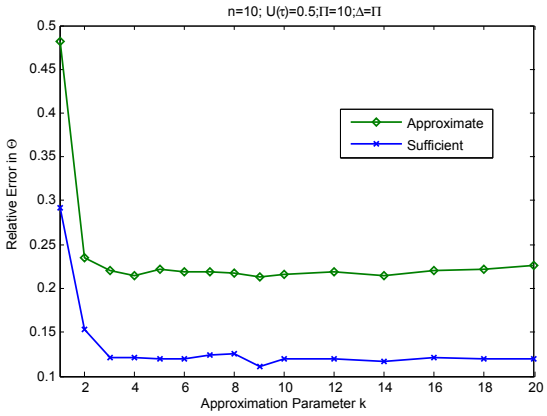


Figure 5.14: Relative Error Vs Approximation

The last plot shown in Figure 5.14 compares the relative error of the approximate and the sufficient algorithm with respect to the exact algorithm. We observe that the relative error for the approximate algorithm (0.48 to 0.23) is almost twice as that of the sufficient algorithm (0.28 to 0.12).

CHAPTER 6: ENFORCEMENT: INTERFACE-BASED MODEL

Demand-curve interfaces provide a precise characterization of the resources demanded by a subsystem using fine-grained approaches such as real-time calculus. In this chapter we focus on an important, unsolved challenge for subsystems specified by such interfaces: the development of efficient enforcement techniques to guarantee temporal isolation between the subsystems. Admission control algorithms can be used in this regard to ensure that the cumulative subsystem demand never violates the demand-curve specified by the interface. To address the problem, we contribute the following in this chapter:

- We propose a simple demand-curve interface model called single-step demand interface (SSDI) and give admission control algorithms for this model with aperiodic workload (Section 6.2).
- For arbitrary demand-curve interface, we propose an exact admission control algorithm for aperiodic jobs. To address the infeasibility of the exact approach, we give an efficient approximate admission control algorithm for this setting (Section 6.3 and 6.7). Further, we verify the efficiency of our admission controllers by simulation over randomly generated job set in Section 6.6.

6.1 Enforcing Demand-Curve Interface for Aperiodic Workload

For the interface enforcement problem, we assume that a subsystem consists of a set of aperiodic jobs as defined in Section 3.1.1, and the demand interface for the subsystem is specified by the definitions in Section 3.3.2. We place no restriction on the parameters of a job in general (except being non-negative numbers) that arrive in the system. In this section, we present some useful functions for characterizing the interaction between the subsystem's generated jobs and the specified interface.

In compositional real-time systems, an interface exposes to the system the temporal requirements of a subsystem. We denote the interface of the subsystem for which we are designing an admission controller as Λ. In this chapter, we assume that Λ is characterized by an SSDI curve (defined in Section 3.3.2) in Section 6.2, and a non-specific demand interface in Section 6.3.

We rephrase the definition of demand-curve interface given in Section 3.3.2. Let T_1 and T_2 represent

discrete time instants ($T_2 \geq T_1$). A demand-bound curve denoted by $\mathsf{dbi}(\Lambda, t)$ for any positive interval $t(= T_2 - T_1)$ of time gives an upper bound on the total demand of the set of jobs J admitted to the subsystem such that the following equation holds.

$$\forall T_1, T_2 \in \mathbb{R} : (0 \leq T_1 < T_2) :: \mathsf{demand}(J, T_1, T_2) \leq \mathsf{dbi}(\Lambda, T_2 - T_1). \tag{6.1}$$

We require that dbi is a right continuous, piecewise linear, non-negative, and non-decreasing function of interval lengths $t \in \mathbb{R}_{\geq 0}$. Note that we denote a time interval as t and an absolute time instant as T.

We now present a function which quantifies how close the subsystem is to exceeding the SSDI curve with respect to all time intervals that end at the last admitted job's deadline.

Definition 16 (Minimum Demand Difference) *The minimum demand difference function $\phi(J, \Lambda, T)$ quantifies the minimum difference between $\mathsf{dbi}(\Lambda, \cdot)$ and $\mathsf{demand}(J, \cdot, \cdot)$ for all intervals ending at some time $T > 0$. If the demand of J is zero for all such intervals, the function returns ∞. More formally,*

$$\phi(J, \Lambda, T) \stackrel{def}{=} \begin{cases} \min\limits_{T' : 0 \leq T' \leq T} \{\psi(J, \Lambda, T', T)\}, & \text{if J} \neq \emptyset; \\ \infty, & \text{if J} = \emptyset. \end{cases} \tag{6.2}$$

where

$$\psi(J, \Lambda, T_1, T_2) \stackrel{def}{=} \mathsf{dbi}(\Lambda, T_2 - T_1) - \mu_0^{-\infty}(\mathsf{demand}(J, T_1, T_2)) \tag{6.3}$$

and $\mu_0^{-\infty}(x)$ equals $-\infty$ if x is zero and x otherwise.

We first prove important statements regarding the ϕ function. These statements will be used to justify and prove the correctness of our proposed admission control algorithms in the subsequent sections. In our first lemma, we show that the ϕ function may be utilized as an exact test of whether a job set satisfies the demand constraints of SSDI interface Λ.

Lemma 36 *For all time instant $T > 0$, $\phi(J, \Lambda, T) \geq 0$, if and only if,*

$$\mathsf{dbi}(\Lambda, T_2 - T_1) \geq \mathsf{demand}(J, T_1, T_2),\ \forall T_1, T_2 \in \mathbb{R} : 0 \leq T_1 < T_2. \tag{6.4}$$

Proof: **(⇒)** If J is empty, the demand is zero for any choice of T_1 and T_2. Thus, since dbi is non-negative for all positive inputs, Equation 6.4 is trivially satisfied. If J is not empty, then $\phi(J, \Lambda, T) \geq 0$ for all

$T > 0$ is equivalent to the following (by the Definition 16).

$$\begin{aligned} &\min_{T':0\leq T'\leq T} \{\psi(J,\Lambda,T',T) \geq 0\}, \quad \forall T > 0 \\ &\Leftrightarrow \psi(J,\Lambda,T',T) \geq 0, \qquad\qquad \forall 0 \leq T' < T \end{aligned}$$

Equation 6.4 follows by substituting in the definition of ψ from Equation 6.3.

(⇐) The other direction of the proof can be obtained by simply reversing the proof above. ■

The next two lemmas (Lemmas 37 and 38) show that for computing ϕ we may restrict attention to time values corresponding to arrivals or deadlines of jobs of J.

Lemma 37 *The value of $\phi(J,\Lambda,T)$ remains unchanged if we restrict in Equation 6.2 the values of T' considered in the* min *function to be from the set $\{A_i \mid j_i \in J\}$.*

Proof: If $J = \emptyset$, the lemma is clearly true. So, let us assume that $J \neq \emptyset$. Let A_0 denote zero and $A_{|J|+1}$ denote T. Consider the partition of the interval $[0,T]$ into subintervals $[A_i, A_{i+1})$ where $0 \leq i \leq |J|$. Assume that the min function in the right-hand side of Equation 6.2 achieves its minimum at some $T' \notin \{A_1, A_2, \ldots, A_{|J|}\}$. Thus, there exists some $i : 0 \leq i \leq |J|$ such that $T' \in (A_i, A_{i+1})$. Now let us consider the time instant A_{i+1}. By Equation 3.1, $\mathsf{demand}(J,T',T)$ is equal to $\mathsf{demand}(J,A_{i+1},T)$ as the set of jobs included in the summation of Equation 3.1 does not change for T' ranging over $(A_i, A_{i+1}]$. Furthermore, since dbi is a non-decreasing function with respect to the interval-length argument and $A_{i+1} > T'$, it must be that $\mathsf{dbi}(\Lambda, T-T') \geq \mathsf{dbi}(\Lambda, T-A_{i+1})$. Thus,

$$\mathsf{dbi}(\Lambda,T-T') - \mathsf{demand}(J,T',T) \geq \mathsf{dbi}(\Lambda,T-A_{i+1}) - \mathsf{demand}(J,A_{i+1},T).$$

Thus, the min function of Equation 6.2 also achieves the same minimum at $A_{i+1} \in \{A_1, A_2, \ldots, A_{|J|}\}$. ■

Lemma 38 *For all time instant $T \geq 0$, the minimum $\phi(J,\Lambda,T)$ occurs at t corresponding to an element of the set $\{\bar{d}_i \mid j_i \in J\}$.*

Proof: The proof is symmetric to the proof of Lemma 37 ■

The next lemma shows how we inductively calculate minimum demand difference ϕ when a new job $j_k \equiv (A_k, E_k, D_k)$ is admitted to the system. To calculate ϕ for job set $J \cup \{j_k\}$, we need to consider

all the intervals that potentially change the value of ϕ upon the arrival of j_k. These intervals are $[A_i, \bar{d}_k]$ where $A_i \leq A_k$ and $[A_k, \bar{d}_i]$ where $\bar{d}_i > \bar{d}_k$, for $\forall j_i \in J$. For all the new intervals with the right endpoint at least $\bar{d}_k$, the demand will increase by E_k. For intervals with right endpoints prior to $\bar{d}_k$, the demand will remain the same. The lemma below describes how to update the minimum demand difference for each of these cases.

Lemma 39 *Given J and j_k such that $A_k \geq \max_{j_i \in J}\{A_i\}$, if $\phi(J, \Lambda, T) \geq 0$ for all $T \in \{\bar{d}_i\}_{j_i \in J}$, then*

$$\phi(J \cup \{j_k\}, \Lambda, T) = \begin{cases} \min \left\{ \begin{array}{l} \mathsf{dbi}(\Lambda, D_k) - E_k, \\ \phi(J, \Lambda, \bar{d}_{last}(J, j_k)) + \sigma(\bar{d}_k - \bar{d}_{last}(J, j_k)) - E_k \end{array} \right\}, & \text{if } T = \bar{d}_k; \\ \min\left\{\phi(J, \Lambda, T) - E_k, \mathsf{dbi}(\Lambda, T - A_k) - E_k\right\}, & \text{if } T > \bar{d}_k; \\ \phi(J, \Lambda, T), & \text{if } T < \bar{d}_k. \end{cases} \tag{6.5}$$

where $\bar{d}_{last}(J, j_k) \stackrel{def}{=} \max_{j_i \in J : \bar{d}_i \leq \bar{d}_k}\{\bar{d}_i\}$. (We assume that $\bar{d}_{last}(J, j_k)$ equals zero if J is empty.)

6.2 Simple Demand-Curve Interface

In this section we propose admission controllers for simple demand-curve interface defined as single-step demand interface SSDI (Equation 3.12). In Section 6.2.1, we present a constant time exact algorithm for MAD aperiodic jobs as subsystem workload and in Section 6.2.2, we present an $O(N)$ time exact algorithm for arbitrary aperiodic jobs as subsystem workload.

6.2.1 Exact Admission Control for MAD Jobs

We start with the simpler case of MAD job arrivals (Section 3.1.1), and present a constant-time admission control algorithm for SSDI interface. Assume at time instant T, n jobs have arrived and been admitted to the subsystem. The MAD property implies that $A_1 + D_1 \leq A_2 + D_2 \leq \cdots \leq A_n + D_n$. Assume that a new job $j_k = (A_k, E_k, D_k)$ arrives in the system with absolute deadline $\bar{d}_k \geq \bar{d}_n$ (Figure 3.1). Job j_k will be accepted if and only if the system can meet total demand over any interval after adding j_k, that is, the demand does not exceed the demand-curve specified by SSDI, $\Lambda \equiv (\sigma, \rho, \nu)$. In our admission controller for MAD jobs, along with checking the demand for new jobs, we keep track of minimum demand difference for future admissions. Our main observation is that we can easily calculate minimum demand difference function at $\bar{d}_k$ by using the last recorded minimum demand difference according Equation 6.5

Algorithm 6 Pseudo-code for admission control of MAD jobs where interface is a single-step demand-curve.

MAD-INITIALIZE()

1 ▷ Let md is the current minimum difference in the system and d is latest absolute deadline among admitted jobs.
2 $\mathsf{d} \leftarrow 0, \mathsf{md} \leftarrow \infty$.

MAD-ADMISSIONCONTROL(j_k)

1 $z \leftarrow \min\left\{\mathsf{dbi}(\Lambda, D_k), \mathsf{md} + \sigma(\bar{\mathsf{d}}_\mathsf{k} - \mathsf{d})\right\} - E_k$
2 **if** $z \geq 0$
3 **Accept** j_k.
4 ▷ Update latest deadline, minimum difference.
5 $\mathsf{d} \leftarrow \bar{d}_k, \mathsf{md} \leftarrow \mathsf{z}$.
6 **else**
7 **Reject** j_k.

of Lemma 39. Furthermore, as there are no active jobs with deadline later than $\bar{d}_k$ at j_k's arrival, we do not need to update the minimum demand difference function at other time values.

The admission control algorithm for MAD jobs is given in Algorithm 6. In this algorithm, at any time point we keep track of two variables: *minimum demand difference* md and *latest absolute deadline* d among admitted jobs in the subsystem. When job j_k arrives we calculate the dbi for the new interval from latest absolute deadline to new job's absolute deadline (the interval will always be greater or equal 0, since the jobs arrive in MAD order). The demand for the new job is its execution E_k. From these two terms we calculate minimum demand difference (z in MAD-ADMISSIONCONTROL) for job j_k and admit the job to the system if z is greater than zero. Finally, if the j_k gets through the admission controller, we update the variables d and md.

Algorithm Complexity

Clearly, MAD-ADMISSIONCONTROL has $O(1)$ time complexity. Each time a new job is admitted to the system, the minimum demand difference is updated for the arriving job only. We do not need to update the minimum demand difference for other jobs as they are unaffected. However, as we will see in the next section, this observation no longer holds for arbitrary job arrival sequences.

Algorithm Correctness

We first give a lemma to show the correspondence between the variable md and the minimum difference ϕ for newly-admitted job j_k.

Lemma 40 *Let $j_1, j_2, \ldots$ be a sequence of admitted jobs that arrive in the subsystem. After the i'th invocation of* MAD-ADMISSIONCONTROL *upon each job arrival,* md *equals $\phi(\{j_1, j_2, \ldots, j_i\}, \Lambda, \bar{d}_i)$,* d *equals $\bar{d}_i$, and $\phi(\{j_1, j_2, \ldots, j_i\}, \Lambda, T) \geq 0$ for all time instant $T > 0$.*

We may now formally show that our admission controller is an exact test for determining whether an admitted job will violate the demand-curve constraints of SSDI Λ.

Theorem 11 MAD-ADMISSIONCONTROL(j_k) *will admit job j_k to a subsystem specified by Λ, if and only if,*

$$\mathsf{demand}(J_{A_k} \cup \{j_k\}, T_1, T_2) \leq \mathsf{dbi}(\Lambda, T_2 - T_1),\ \forall 0 \leq T_1 \leq T_2. \tag{6.6}$$

where J_{A_k} is the set of admitted jobs upon j_k's arrival.

Proof: **(⇐)** We prove this direction by contrapositive. Thus, assume that j_k is rejected by MAD-ADMISSIONCONTROL(j_k). By Lemma 40, the variable md corresponds to $\phi(J_{A_k}, \Lambda, \bar{d}_{|J_{A_k}|})$ and $\phi(J_{A_k}, \Lambda, T)$ for all time instant $T > 0$ at the beginning of the invocation of MAD-ADMISSIONCONTROL(j_k). Thus, by Lemma 39, $\phi(J_{A_k} \cup \{j_k\}, \Lambda, \bar{d}_k)$ equals

$$\min \left\{ \begin{array}{l} \mathsf{dbi}(\Lambda, D_k) - E_k, \\ \phi(J_{A_k}, \Lambda, \bar{d}_{|J_{A_k}|}) + \sigma(\bar{d}_k - \bar{d}_{|J_{A_k}|}) - E_k \end{array} \right\}.$$

Since j_k is rejected, it must have failed the condition of Line 2. This implies the above expression is less than zero. For either value of the minimum, it may easily be shown that this implies the negation of Equation 6.6 of the lemma.

(⇒) This direction follows immediately from Lemma 40. ■

6.2.2 Exact Admission Control for Arbitrary Jobs

In this section we relax the constraint of the MAD property for the aperiodic jobs in the system; that is, jobs may arrive in the system at any order of deadline (Figure 3.2). For each new job arrival, the admission

controller must check the demand of all preceding and succeeding jobs considering the execution requirement of the new job. For admission control purposes, a data structure containing the admitted jobs needs to be maintained. The data structure we propose in this thesis is a variant of the sorted data structures proposed by Andersson and Ekelin [9] and Lipari and Baruah [62].

Algorithm 7 Pseudo-code for admission control of aperiodic jobs where interface is a single-step demand-curve.

INITIALIZE()

1 ▷ Let $\mathbb{L}$ represent list of nodes. Each node $\mathcal{P}_i \in \mathbb{L}$ comprise parameters: arrival time A, execution time E and absolute deadline d of an admitted job, two variables pd, sd, and a pointer next to node. $\mathbb{L}$ is sorted in deadline order.

2 $\mathcal{P}_0 \equiv (0, 0, 0, \infty, \infty, \mathsf{null}), \mathcal{P}_\infty \equiv (0, \infty, 0, \infty, \infty, \mathsf{null})$

3 $\mathcal{P}_0.\mathsf{next} \leftarrow \mathcal{P}_\infty$

UPONJOBARRIVAL(j_k)

1 ▷ Let $\mathcal{P}_\ell.d \leq \bar{d}_k \leq \mathcal{P}_{\ell+1}.d$

2 $\mathsf{pd} \leftarrow \min\left\{\mathcal{P}_\ell.\mathsf{pd} + \sigma(\bar{\mathsf{d}}_\mathsf{k} - \mathcal{P}_\ell.\mathsf{d}), \mathsf{dbi}(\Lambda, \mathsf{D_k})\right\} - \mathsf{E_k}$

3 $\mathsf{sd} \leftarrow \min\left\{\mathcal{P}_{\ell+1}.\mathsf{pd}, \mathcal{P}_{\ell+1}.\mathsf{sd}, \mathsf{dbi}(\Lambda, \mathcal{P}_{\ell+1}.\mathsf{d} - \mathsf{A_k})\right\} - \mathsf{E_k}$

4 **if** $\mathsf{pd} \geq 0$ and $\mathsf{sd} \geq 0$

5 **Accept** j_k.

6 Allocate new node $\mathcal{P}_i \equiv (A_k, \bar{d}_k, E_k, \mathsf{pd}, \mathsf{sd}, \mathsf{null})$, and insert to $\mathbb{L}$ in order.

7 $\mathcal{P}_i.\mathsf{next} \leftarrow \mathcal{P}_\ell.\mathsf{next}, \mathcal{P}_\ell.\mathsf{next} \leftarrow \mathcal{P}_i$

8 For all $\ell(\neq 0) < i$, set $\mathcal{P}_\ell.\mathsf{sd} \leftarrow \min\left\{\mathcal{P}_\ell.\mathsf{sd}, \mathcal{P}_\mathsf{i}.\mathsf{sd}, \mathcal{P}_\mathsf{i}.\mathsf{pd}\right\}$.

9 For all $\ell(\neq \infty) > i$, set $\mathcal{P}_\ell.\mathsf{pd} \leftarrow \min\left\{\mathcal{P}_\ell.\mathsf{pd}, \mathsf{dbi}(\Lambda, \mathcal{P}_\ell.\mathsf{d} - \mathcal{P}_\mathsf{i}.\mathsf{A})\right\} - \mathcal{P}_\mathsf{i}.\mathsf{E}$ and $\mathcal{P}_\ell.\mathsf{sd} \leftarrow \min\left\{\mathcal{P}_\ell.\mathsf{sd}, \mathsf{dbi}(\Lambda, \mathcal{P}_{\ell+1}.\mathsf{d} - \mathcal{P}_\mathsf{i}.\mathsf{A})\right\} - \mathcal{P}_\mathsf{i}.\mathsf{E}$.

10 **else**

11 **Reject** j_k.

UPONJOBDEADLINE(j_k)

1 $\mathcal{P}_i \leftarrow \mathcal{P}_0.\mathsf{next}$

2 Set $\mathcal{P}_0.d = \mathcal{P}_i.d$ and $\mathcal{P}_0.\mathsf{pd} = \mathcal{P}_\mathsf{i}.\mathsf{pd}$.

3 Delete $\mathcal{P}_i$ from $\mathbb{L}$.

In Algorithm 7, we propose admission controller for arbitrary aperiodic jobs (i.e., no constraint on arrivals/deadlines). The linked-list data structure $\mathbb{L}$ stores, for each admitted, active job $j_k \equiv (A_k, E_k, D_k)$ in the subsystem, a node $\mathcal{P}_i \equiv (A, E, d, \mathsf{pd}, \mathsf{sd}, \mathsf{next})$ where A corresponds to arrival time A_k of j_k, E corresponds to execution requirement E_k of j_k, d corresponds to absolute deadline $\bar{d}_k$ of j_k, the variable pd represents *preceding minimum demand difference*, the variable sd represents *successive minimum demand*

difference, and next is a pointer to node. The variable $\mathcal{P}_i$.pd keeps track of the minimum demand difference at time $\mathcal{P}_i.d$. The variable $\mathcal{P}_i$.sd stores the minimum demand difference for the jobs with deadlines after $\mathcal{P}_i.d$ (i.e., the pd values for the succeeding nodes in the sorted list $\mathbb{L}$). The list is initialized by INITIALIZE() to contain two dummy nodes in order of deadline: a leftmost node $\mathcal{P}_0 \equiv (0, 0, 0, \infty, \infty, \mathcal{P}_\infty)$ and a rightmost node $\mathcal{P}_\infty = (0, 0, \infty, \infty, \infty, \mathsf{null})$.

When a new job j_k arrives in the system, UPONJOBARRIVAL(j_k) in Algorithm 7 finds the appropriate insertion position (according to the absolute deadline $\bar{d}_k$) in the list in $O(N)$ time. Assume it is between the nodes $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ (i.e., $\mathcal{P}_\ell.d \leq \bar{d}_k \leq \mathcal{P}_{\ell+1}.d$). The variable pd for job j_k is calculated from $\mathcal{P}_\ell$'s preceding minimum demand difference ($\mathcal{P}_\ell$.pd), and the difference in demand and demand interface for this job (similar to md of MAD-ADMISSIONCONTROL in Algorithm 6). The variable sd is calculated by taking minimum of the $\mathcal{P}_{\ell+1}$'s preceding minimum demand difference($\mathcal{P}_{\ell+1}$.pd) and successive minimum demand difference ($\mathcal{P}_{\ell+1}$.sd). The new job j_k is admitted if both pd and sd values are greater than 0. Intuitively, the condition pd $\geq$ 0 determines that if j_k is added, than its deadline will be met by the demand interface dbi and the condition sd $\geq$ 0 ensures that after adding j_k, all the succeeding jobs in the system with deadline later than $\bar{d}_k$ (nodes in the list succeeding the node $\mathcal{P}_\ell$) will also meet their deadline. In this way, the algorithm ensures the demand of the system is met by the SSDI curve.

If UPONJOBARRIVAL(j_k) returns "accept", a new node $\mathcal{P}_i \equiv (A_k, E_k, \bar{d}_k, \mathsf{pd}, \mathsf{sd}, \mathsf{null})$ corresponding to j_k is created and inserted to $\mathbb{L}$ between $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$. The demand for the succeeding nodes is increased by the amount of the new job's execution E_k. Thus, the preceding minimum difference (pd) and succeeding minimum difference (sd) values for nodes succeeding $\mathcal{P}_i$ in the sorted list must also be reduced by E_k. The demand for preceding nodes of $\mathcal{P}_i$ is not increased, since they have deadlines before $\bar{d}_k$, which implies that the pd values for the nodes preceding $\mathcal{P}_i$ in the sorted list will remain unchanged. However, the successive minimum difference for the preceding nodes should be updated, if this is less than $\mathcal{P}_i$.sd or $\mathcal{P}_i$.pd, to reflect the change in minimum difference of successive nodes after inserting $\mathcal{P}_i$. When UPONJOBARRIVAL(j_k) returns "reject", the job j_k is not admitted to the system and $\mathbb{L}$ remains unchanged.

Algorithm Complexity

Algorithm 7 uses a simple linked-list implementation which requires $O(N)$ time for determining whether a job should be admitted where N is an upper bound on the number of active jobs in the subsystem at any given time. The updates to both the preceding and succeeding jobs requires $O(N)$. When the deadline of a job is elapsed, it can be removed from the data structure in $O(1)$ time according to UPONJOBDEADLINE(j_k) since the nodes are ordered by absolute deadline. This data structure can be enhanced by using balanced-binary tree, and lazy update operation can be performed similar to Andersson and Ekelin [9] to reduce the time complexity for admission control to $O(\log N)$.

Algorithm Correctness

The functionality of the algorithm is similar to the approach proposed by Andersson and Ekelin [9] for uniprocessor interface (i.e., the demand-curve is specified by the function $\mathsf{dbi}(t) = t$) for a system with set of aperiodic and periodic jobs as workload. Thus we omit the proof of correctness for this algorithm.

6.3 Arbitrary Demand-Curve Interface

We now focus on enforcing arbitrary demand-curve interface for a subsystem of a compositional real-time system. In Chapter 2, we described several real-time interface models that have been proposed in recent years (e.g., the EDP resource model Ω, real-time calculus [88], etc.); however, we consider Λ to be from a non-specific interface model in this section. Our only requirement is that the interface model permits a characterization of the admissible demand over intervals of time. Such interface can be modeled as sum of multiple SSDI curves as shown in Section 3.3.2. Throughout this section, we assume that the system designer has already generated and specified the interface Λ. The challenge of generating and composing demand-curve interfaces is important, but orthogonal to the problem we address here. (See Thiele et al. [88] for a discussion of these issues.)

Unlike the single-step demand-curve interface, the enforcement for arbitrary demand-curve interface is a complex problem and the solution can become infeasible for long running online systems. As a starting point of our development, we restrict ourselves to MAD aperiodic jobs for this setting in next four sections and then give a straightforward extension for non-MAD aperiodic jobs in Section 6.7. For

arbitrary demand-curve interface, in Section 6.3.1, we present an $O(n)$ time exact algorithm for MAD aperiodic jobs as subsystem workload, where n is the total number of jobs arrived in the subsystem from system startup time. In Section 6.3.2 we show via an example that the algorithm becomes infeasible over time, and to address this problem, we develop an approximate admission control algorithm independent of the number of jobs in the subsystem in Section 6.3.3. Developing admission controller for arbitrary aperiodic jobs as subsystem workload is addressed in Section 6.7.

6.3.1 Exact Admission Control for MAD Jobs

Let us consider the scenario where we have already admitted a set of MAD jobs J in the system and are attempting to determine whether we can admit a new job j_k (where j_k has later arrival time and deadline than all previously-admitted jobs). Observe that an exact admission control algorithm is conceptually relatively straightforward: to check whether a job j_k can be admitted, calculate the change in demand over every interval $[T_1, T_2]$ and check the inequality of Equation 6.1. However, a naive implementation of this idea would require the evaluation of Equation 6.1 for an infinite number of intervals. A practical (finite-time) implementation of the exact algorithm can be developed from the observation that only a finite number of intervals must be checked for determining whether to admit j_k: intervals that begin at the arrival of some job of J and end at $\bar{d}_k$.

The exact algorithm shown in EXACTAC and EXACTAC-INIT (Algorithm 8) maintains an ordered set S of intervals each specified by a demand pair (x, y) where x corresponds to the length of the interval and y corresponds to demand over that interval. The set S is ordered in non-decreasing value of the x-coordinate for each pair. The variable $\bar{d}_{last}$ stores last accepted job's absolute deadline. The demand pair can be mapped to a point in the cartesian plane (Figure 6.1). Furthermore, we will show later that it is only necessary to store demand pairs in S where the interval length x corresponds to the difference between the arrival time of some accepted job of J and $\bar{d}_{last}$ as shown in Figure 6.1.

When a new job j_k arrives in the system, the demand of the intervals $(\bar{d}_k - A_i), \forall j_i \in J$ needs to be checked. In Line 2 of EXACTAC, an (x, y) interval corresponding to j_k is added to the set. Since jobs arrive in MAD order, the intervals ending at $\bar{d}_k$ can be obtained by incrementing x values in the set S by $\delta_x = \bar{d}_k - \bar{d}_{last}$, where $\bar{d}_{last}$ is the absolute deadline of the last admitted job before j_k, and the demand over these intervals can be obtained by incrementing y values in the set by $\delta_y = E_k$ amount. For all the new

Algorithm 8 Pseudo-code for exact admission control of MAD jobs where interface is an arbitrary demand-curve.

EXACTAC-INIT()

▷ S (initially empty) will keep a set of (x, y) values.
1 $S \leftarrow \emptyset$
2 $\bar{d}_{last} \leftarrow 0$

EXACTAC(Λ, j_k)

1 $\delta_x \leftarrow \bar{d}_k - \bar{d}_{last}; \delta_y \leftarrow E_k$
▷ Insert a point in S corresponding to j_k.
2 $S \leftarrow S \cup \{(D_k - \delta_x, E_k - \delta_y)\}$
3 **for** $(x, y) \in S$
4 **if** dbi$(\Lambda, x + \delta_x) < (y + \delta_y)$
▷ Delete previously inserted point.
5 $S \leftarrow S \setminus \{(D_k - \delta_x, E_k - \delta_y)\}$
6 **Reject** j_k.
7 **end for**
8 **Accept** j_k.
▷ Shift points in x, y-plane.
9 **for** $(x, y) \in S$
10 $S \leftarrow S \setminus \{(x, y)\}$
11 $S \leftarrow S \cup \{(x + \delta_x, y + \delta_y)\}$
12 **end for**
13 $\bar{d}_{last} \leftarrow \bar{d}_k$

intervals with the increment, the algorithm checks if the demand in the interval is less than the dbi (Line 3 to Line 7). If for any interval this condition is violated, j_k is rejected and the point corresponding to it is removed from S. When the condition holds for all the points in S, j_k is admitted and all the points in S are updated by the incrementing (δ_x, δ_y) amount (Line 9 to Line 12). Figure 6.1 illustrates the algorithm after j_4 arrives to the system. The set S consists of the intervals ending at $\bar{d}_3$ (blue points shown in the plot) prior to the arrival of j_4. When j_4 arrives, after successful admission test, the demand pair $(\bar{d}_4 - A_4, E_4)$ is added (red point in the figure) in S, and all the existing intervals are shifted by $(\delta_x = \bar{d}_4 - \bar{d}_3, \delta_y = E_4)$ amount.

Algorithm Complexity

Exact admission control for arbitrary demand-curve is computationally linear to the number of jobs that have arrived in the system. Since all past intervals need to be verified, the number of such intervals

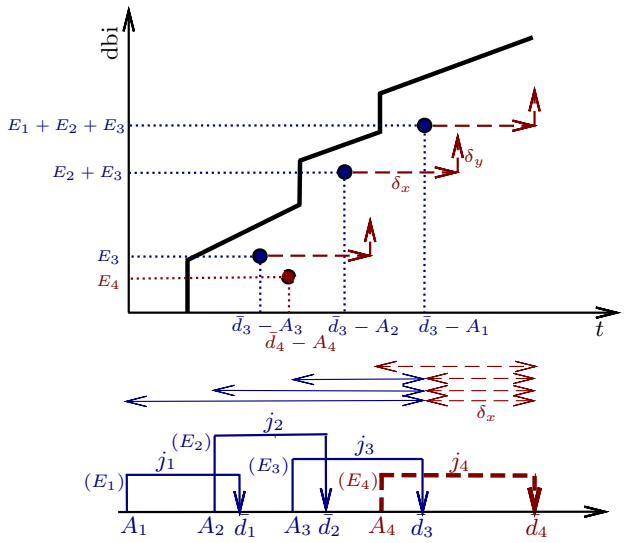


Figure 6.1: Illustration of exact admission control algorithm.

become intractable with time. The approach used for single-step demand interface [36] (i.e., dbi is linear) in Section 6.2 is not applicable to arbitrary demand curves. The main challenge for arbitrary demand interface is that the difference in demand and the interface for any time-interval changes non-linearly over time. More specifically, for a same increase in interval length and same increase in demand, the interval which had minimum demand difference before will not necessarily remain *minimum* difference after the increase.

Algorithm Correctness

We now provide the main theorem which states that EXACTAC is correct and exact.

Theorem 12 *Given a set of previously-admitted jobs J, the procedure* EXACTAC(Λ, j_k) *returns "Accept", if and only if, j_k may be admitted without $J \cup \{j_k\}$ violating Λ.*

We need to prove the next three lemmas in order to prove the theorem. Please refer to Appendix A for details proof of the lemmas. The first lemma shows that we only need to check a finite number (with respect to the number of jobs in J) to determine whether Equation 6.1 is satisfied.

Lemma 41 *For any set of* MAD *jobs J, Equation 6.1 is true, if and only if,*

$$\forall j_i, j_\ell \in J : \bar{d}_i \leq \bar{d}_\ell :: \mathsf{demand}(J, A_i, \bar{d}_\ell) \leq \mathsf{dbi}(\Lambda, \bar{d}_\ell - A_i). \tag{6.7}$$

The next lemma shows the correspondence between points stored in S and the intervals ending at the deadline of the last admitted job (i.e., $\bar{d}_{last}$).

Lemma 42 *After the call to* EXACTAC-INIT() *and the k'th invocation of* EXACTAC(Λ, j_k) *where $k \in \mathbb{N}$, for* MAD*-sequenced jobs $j_1, j_2, \ldots, j_k$, define J_k to be the set of jobs admitted by the algorithm. For each job $j_\ell \in J_k$, there exists $(x, y) \in S$ such that x equals $\bar{d}_{last} - A_\ell$ and y equals* demand$(J_k, A_\ell, \bar{d}_{last})$. *Furthermore, $\bar{d}_{last}$ equals* $\max\left\{\{0\} \cup \{\bar{d}_\ell \mid j_\ell \in J_k\}\right\}$.

Finally, to determine whether to admit j_k, we only need to check intervals ending at $\bar{d}_k$ (and that begin at some arrival time). Since we have stored all intervals that end at $\bar{d}_{last}$ ($\leq \bar{d}_k$) in S, we may update the points in S by shifting them to the right $\delta_x = \bar{d}_k - \bar{d}_{last}$ and upwards by $\delta_y = E_k$. We also add a point (D_k, E_k) which corresponds to job j_k's demand over its arrival and deadline. A new or newly-shifted point will be above dbi$(\Lambda, \cdot)$, if and only if, the corresponding interval ending at $\bar{d}_k$ violates Equation 6.1. This observation is formalized in the next lemma; thus, the algorithm returns "Accept", if and only if, it is safe to do so.

Lemma 43 *After the call to* EXACTAC-INIT() *and the k'th invocation of* EXACTAC(Λ, j_k) *where $k \in \mathbb{N}$, for* MAD*-sequenced jobs $j_1, j_2, \ldots, j_k$, define J_k to be the set of jobs admitted by the algorithm. It must be that Equation 6.1 holds for J_k and Λ. Furthermore, if job j_k was rejected by the admission controller (i.e.,* EXACTAC(Λ, j_k) *returns "reject" and $j_k \notin J_k$), then Equation 6.1 is false for $J_{k-1} \cup \{j_k\}$.*

The combination of Lemmas 41, 42, and 43 immediately imply Theorem 12.

6.3.2 Inadequacy of Naive Reduction in Intervals

A natural idea for reducing the time complexity of the exact admission controller would be to discard some of the points of S. Unfortunately, we will see that this idea will lead to an incorrect admission controller. In this subsection, we give a job set J such that if any of the points in S (corresponding jobs in J) is discarded, there exist future job arrivals that will cause a violation of Equation 6.1, but the admission controller will not detect this violation.

Example 1 Consider a system that has admitted a set of jobs $J = \{j_1, \ldots j_4\}$ as shown in Figure 6.2(a). The demand-interface and the intervals corresponding to points in S (shown in grey) are illustrated in

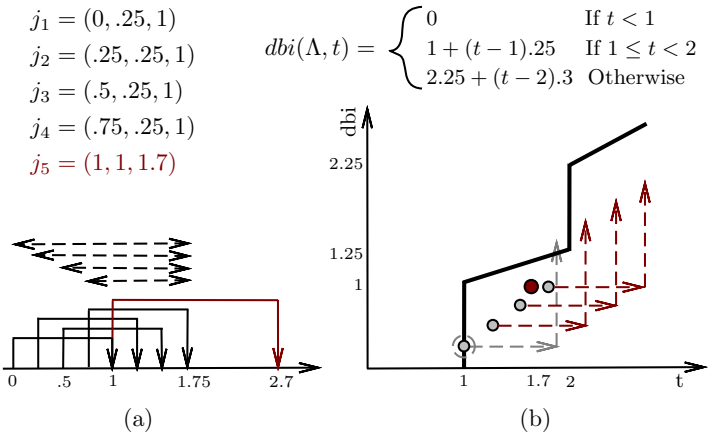


Figure 6.2: Naive reduction in intervals might cause violation of dbi.

Figure 6.2(b). For any of the points in S, if we discard it, we can construct a new job with parameters such that it would have caused the discarded point to exceed the dbi after shifting it towards right ($\delta_x = \bar{d}_k - \bar{d}_{last}$) and upwards ($\delta_y = E_k$). We will show that for the dbi and each of the intervals in Figure 6.2, a job can be constructed with parameters such that only one interval violates the dbi after the shift. In other words, if we discard one interval, the demand-interface violation might be undetected for the new job, which will lead to an incorrect admission controller.

First, consider that we have discarded the point corresponding to $[A_4, \bar{d}_4] = [.75, 1.75]$ (the point within circle in Figure 6.2(b)). The point corresponding to this interval has the largest difference with dbi after the arrival of j_4. Let the next job, $j_5(A_5, E_5, D_5) \equiv (1, 1, 1.7)$, arrive into the system. The demand for the new interval $[1, 2.7]$ is 1, thus the point corresponding to this interval is below the dbi. Now all the other points $[A_i, \bar{d}_5], i = 1 \ldots 3$ will be shifted by $(\delta_x, \delta_y) \equiv (2.7 - 1.75, 1)$, and they are safely below the dbi. But the point corresponding to $[A_4, \bar{d}_4]$ interval would have moved beyond the dbi as $\text{dbi}(1 + 0.95) = 1.2375 \le 0.25 + 1$. Since this point is discarded, the system will admit j_5, which will eventually lead to a violation of demand curve interface. Similarly, if we discard $[A_3, \bar{d}_4]$ interval, and insert a new job $j_5 = (1, 0.7, 1.5)$, the demand for the discarded interval will exceed dbi only, and other intervals will stay below dbi after the shift. Again, let us assume that $[A_2, \bar{d}_4]$ interval is discarded, and a new job arrives with parameters $j_5 = (1, 0.5, 1.2)$, then the increment in all the other intervals will not violate the dbi, and only demand for $[A_2, \bar{d}_4]$ interval will violate the dbi. Finally, consider we have discarded the $[A_1, \bar{d}_4]$ interval and a new job $j_5 = (1, 0.25, 0.95)$ arrives in the system. Here also all the other intervals with their increment will satisfy the dbi and falsely accept j_5 to the system, whereas if we

have stored the $[A_1, \bar{d}_4]$ interval, the interface violation would have been detected. ■

This example illustrates that there exist sequences of job arrivals and dbi such that no point may naively be discarded. Therefore, simply throwing away intervals is not a solution to reducing the complexity of the admission controller.

6.3.3 Approximate Admission Control for MAD Jobs

As mentioned in the previous section, the exact admission controller needs to check every demand-point corresponding to previously admitted jobs while admitting a new job in the system. Thus the algorithm is intractable for long-running online systems. In this section we propose an approximate solution to efficiently perform admission control for MAD jobs. In our proposed approach, we reduce the number of intervals (points) using a more sophisticated approach than just naively dropping intervals as illustrated in the example of Section 6.3.2. We achieve our reduction in the time complexity for admission control via four main steps:

1. *Divide the xy-Plane into Regions:* As a first step towards reducing the number of stored intervals, we divide the xy-plane into increasingly large intervals based on a user-supplied accuracy parameter $\epsilon > 0$. A smaller value of ϵ will indicate that the admission controller is more accurate (i.e., closer to the exact admission controller); however, the time complexity of the algorithm will be increased.

Definition 17 ($(1+\epsilon)$-Region) *The i'th $(1+\epsilon)$-region denoted by $\mathcal{A}^i$ for $i \in \mathbb{N}^+$ is a horizontal strip in Euclidean space (i.e., $\mathbb{R}^2$) where the upper boundary is $(1+\epsilon)$ times the lower boundary of $\mathcal{A}^i$. Formally, the i'th $(1+\epsilon)$-region is defined as*

$$\mathcal{A}^i \stackrel{def}{=} \left\{(x,y) \in \mathbb{R}^2 \mid (1+\epsilon)^{i-1} \leq y < (1+\epsilon)^i\right\} \tag{6.8}$$

Figure 6.3 gives a visual depiction of the horizontal division of the xy-plane. We denote the lower bound and upper bound of region $\mathcal{A}^i$ as $\mathcal{A}^i.lb$ and $\mathcal{A}^i.ub$ respectively.

2. *Merge Intervals Within A Region:* Consider two distinct intervals represented by (x_l, y_l) and (x_r, y_r) in the same $(1+\epsilon)$-region $\mathcal{A}^i$. As a method of reducing the number of intervals stored, we will merge two such intervals into an *approximation point*.

Definition 18 (Approximation Point $\hat{P}\left((x_l, y_l), (x_r, y_r)\right)$) *Consider two points (x_l, y_l) and (x_r, y_r) where*

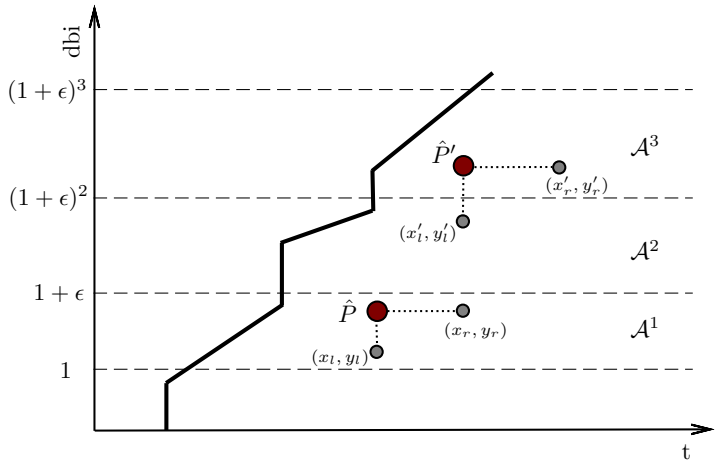


Figure 6.3: Approximating Y-axes of dbi.

$x_l \leq x_r$ *and* $y_l \leq y_r$. *We define the* approximation point $\hat{P}\left((x_l, y_l), (x_r, y_r)\right)$ *"anchored" by points* (x_l, y_l) *and* (x_r, y_r) *as*

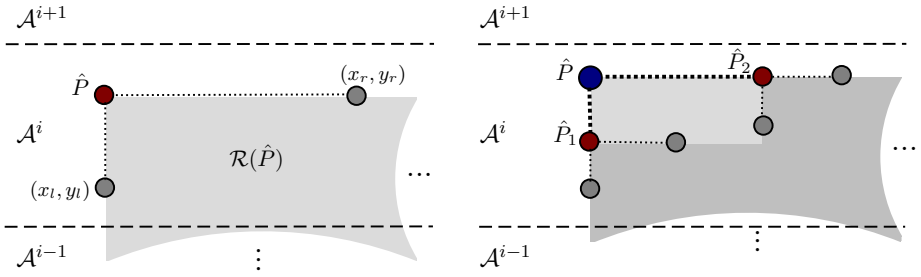
$$\hat{P}\left((x_l, y_l), (x_r, y_r)\right) \stackrel{def}{=} (x_l, y_r). \tag{6.9}$$

The points (x_l, y_l) *and* (x_r, y_r) *are referred to as the* left-anchor point *and* right-anchor point *of approximation point* $\hat{P}$.

For simplicity, we drop the "$((x_l, y_l), (x_r, y_r))$" from $\hat{P}$ when it is clear which approximation point we are referring to. The notation $\hat{P}.x_l$ and $\hat{P}.y_l$ (respectively, $\hat{P}.x_r$ and $\hat{P}.y_r$) refers to the x and y coordinates of the left (right) anchor point. Figure 6.4(a) shows the formation of the approximation point $\hat{P}$ from its two anchor points. One important point to observe is that, due to the fact that we merge points in the same $(1+\epsilon)$-region, it must be that $\hat{P}.y_r \leq (1+\epsilon)\hat{P}.y_l$. In other words, the y-value of the approximation point is no more than a factor of $(1+\epsilon)$ greater than the y-value of its left anchor point. This observation will be useful in proving the approximation ratio of our admission controller. In the next step, we show how these approximation points can be utilized to decrease the total number of intervals stored by our approximate admission controller.

3. *Eliminate Redundant Points:* Observe from Figure 6.4(a) that the region below and to the right of the approximation point $\hat{P}$ forms a rectangular region extending infinitely to the right and below, called the *redundancy region* for approximation point $\hat{P}$.

Definition 19 (Redundancy Region) *A* redundancy region *for approximation point* $\hat{P}$ *is the region ex-*



(a) Redundancy Region (b) Merge of Points

Figure 6.4: Approximation point $\hat{P}$ in the $(1+\epsilon)$-region $\mathcal{A}^i$.

tending towards lower right of a point in cartesian plane:

$$\mathcal{R}(\hat{P}) \stackrel{def}{=} \left\{(x,y) \in \mathbb{R}^2 | (\hat{P}.x_l \leq x) \wedge (\hat{P}.y_r \geq y)\right\}. \tag{6.10}$$

The following observation allows us to ignore intervals corresponding to points that fall into this rectangular region; we call points falling into this region *redundant points*.

Lemma 44 *For a given point $\hat{P}$ and any point $(x,y) \in \mathbb{R}^2$, if $x \geq \hat{P}.x_l$ and $y \leq \hat{P}.y_r$ then* $\mathsf{dbi}(\Lambda, \hat{P}.x_l) - \hat{P}.y_r \leq \mathsf{dbi}(\Lambda, x) - y$.

Proof: Since dbi is a non-decreasing function, $\mathsf{dbi}(\Lambda, x) \geq \mathsf{dbi}(\Lambda, \hat{P}.x_l)$. Combining this with the condition $\hat{P}.y_r \geq y$, we obtain $\mathsf{dbi}(\Lambda, \hat{P}.x_l) - \hat{P}.y_r \leq \mathsf{dbi}(\Lambda, x) - y$. ■

Therefore, we can conclude that if approximate point $\hat{P}$ is below dbi, then any point in the redundancy region $\mathcal{R}(\hat{P})$ will also be below dbi.

4. *Merge Approximation Points:* In Step 2, we show how two intervals in the same $(1+\epsilon)$-region can be merged into a single approximation point. We will see in the next subsection that each approximation point is shifted to the right and upwards as new jobs are admitted to the system. Thus, an approximation point formed in region $\mathcal{A}^i$ may eventually move (completely with both anchor points) into another region $\mathcal{A}^j$ where $j > i$. We say that an approximation point $\hat{P}$ is "completely in" $\mathcal{A}^j$ if $\mathcal{A}^j.lb \leq \hat{P}.y_l \leq \hat{P}.y_r \leq \mathcal{A}^j.ub$; otherwise, the approximation point "straddles" $(1+\epsilon)$-regions. An approximation point $\hat{P}$ is "contained" (not necessarily completely) in $\mathcal{A}^j$, if $\mathcal{A}^j.lb \leq \hat{P}.y_r \leq \mathcal{A}^j.ub$. Consider any two approximation points $\hat{P}_1$ and $\hat{P}_2$ that are completely in $\mathcal{A}^j$; i.e., for $k \in \{1,2\}$, the approximation points $\hat{P}_k$ satisfies $\hat{P}_k.y_l \geq \mathcal{A}^j.lb$ and $\hat{P}_k.y_r \leq \mathcal{A}^j.ub$. Given these two points, we may merge $\hat{P}_1$ and $\hat{P}_2$ to form

a new approximation point $\hat{P} = \left(\min_{k \in \{1,2\}} \{\hat{P}_k.x_l\}, \max_{k \in \{1,2\}} \{\hat{P}_k.y_r\} \right)$. Figure 6.4(b) depicts the merge of two approximation points and evolution of their redundancy regions after merging.

In Algorithm 9, we present the pseudocode for our approximate admission control algorithm for an arbitrary demand-curve interface. The algorithm keeps a linked list of nodes $\mathbb{L}$, where each node represents an approximation point corresponding to a $(1+\epsilon)$-region. Each node $\hat{P}$ consists of two points: left anchor (x_l, y_l) and right anchor (x_r, y_r), and a pointer next to the next approximation point. We abuse notation somewhat to allow $\hat{P}$ to refer to both the point in the xy-plane and the node in the list $\mathbb{L}$; $\hat{P}$.next is the next node after $\hat{P}$ in the list or null if no such node exists. The nodes of $\mathbb{L}$ are ordered in increasing value of x_l. The list is initially empty. At any time the algorithm keeps a variable $\bar{d}_{last}$ to store the last admitted job's deadline.

First we describe some helper subroutines used by the approximate admission controller shown in Algorithm 9. The procedure APPROXIMATEAC-INIT() initializes the list $\mathbb{L}$ to empty, $\bar{d}_{last}$ to zero and $\hat{P}_h$ to null. An INSERT(x, y) operation creates a node in the list $\mathbb{L}$ with the approximate point equal to the anchor points; i.e., $(x_l, y_l) = (x_r, y_r) = (x, y)$. The node is inserted into the list in non-decreasing order of its x_l-value. A DELETE$(\hat{P})$ operation deletes node $\hat{P}$ from the list. A SHIFT$(\hat{P}, \delta_x, \delta_y)$ operation shifts both the left anchor and the right anchor of $\hat{P}$ by (δ_x, δ_y) amount, where δ_x is the shift in X-axes and δ_y is the shift in Y-axes. The MERGE$(\hat{P}_1, \hat{P}_2)$ operation as described earlier in this section merges two nodes by updating the left anchor and right anchor of $\hat{P}_1$ with the left-most of the two left anchors, and the top-most of the two right anchors respectively (Figure 6.4(b)). Then it deletes the node $\hat{P}_2$ and updates the linked list accordingly.

When a new job $j_k(A_k, E_k, D_k)$ arrives in the system, APPROXIMATEAC first checks in Line 1 whether the demand E_k of the job over the interval D_k exceeds the demand interface over a D_k-length interval, i.e., $\mathsf{dbi}(\Lambda, D_k)$. It is possible to show (similar to Lemma 42) that all the intervals of interest for MAD jobs end at the last admitted job's deadline $\bar{d}_{last}$. Using this fact, the potential increase (due to accepting j_k) in the interval lengths is $\delta_x = \bar{d}_k - \bar{d}_{last}$ and the increase in demand is $\delta_y = E_k$. In the *while*-loop of Lines 7 to 15, we check that each of the approximate points will still fall below the $\mathsf{dbi}(\Lambda, \cdot)$ if they are shifted to the right by δ_x and upwards by δ_y. In particular, Line 8 determines if the individual points shifted will violate Λ; Line 12 determines if any new approximation point, formed by the shift (due to the merging described in Step 4 of the previous subsection), will violate the interface Λ. If a violation

is detected, j_k will be rejected; otherwise, we may accept the job.

Once the conditions are verified for all the nodes in the list, the algorithm "accepts" the job and performs UPDATE$(\hat{P}_h, \delta_x, \delta_y)$ operation for all nodes starting from the head of the list. This procedure shifts each node (both left and right anchors) by (δ_x, δ_y) amount using SHIFT and performs MERGE operation when two consecutive nodes are in the same $(1+\epsilon)$-region.

Algorithm 9 Pseudo-code for approximate admission control of MAD jobs where interface is an arbitrary demand-curve.

APPROXIMATEAC-INIT()

1 ▷ Each node $\hat{P}$ in the list $\mathbb{L}$ consists of two points $(x_l, y_l), (x_r, y_r)$, and a pointer next.
2 ▷ Initially $\mathbb{L}$ is empty. Let $\hat{P}_h$ represents the head of the list
3 $\hat{P}_h \leftarrow$ null; $\bar{d}_{last} \leftarrow 0$

APPROXIMATEAC(Λ, j_k, ϵ)

1 **if** dbi$(\Lambda, D_k) < E_k$
2 **Reject** j_k.
3 $\delta_x \leftarrow \bar{d}_k - \bar{d}_{last}; \delta_y \leftarrow E_k$
▷ Insert node at the beginning of the list
4 $\hat{P}_c \leftarrow$ INSERT$(D_k - \delta_x, E_k - \delta_y)$
5 $\hat{P}_c$.next $\leftarrow \hat{P}_h; \hat{P}_h \leftarrow \hat{P}_c$
6 $\hat{P}_p \leftarrow \hat{P}_h; \hat{P}_c \leftarrow \hat{P}_h$.next
7 **while** $\hat{P}_c$ not null
8 **if** dbi$(\Lambda, \hat{P}_c.x_l + \delta_x) < \hat{P}_c.y_r + \delta_y$
9 Delete the head of the list. **Reject** j_k.
▷ $\hat{P}_p$ and $\hat{P}_c$ would have moved to same region
10 $j \leftarrow \lceil \log_{1+\epsilon}(\hat{P}_p.y_l + \delta_y) \rceil; k \leftarrow \lceil \log_{1+\epsilon}(\hat{P}_c.y_r + \delta_y) \rceil$
11 **if** $j == k$
12 **if** dbi$(\Lambda, \min\{\hat{P}_p.x_l, \hat{P}_c.x_l\} + \delta_x) < (\max\{\hat{P}_p.y_r, \hat{P}_c.y_r\} + \delta_y)$
13 Delete the head of the list. **Reject** j_k.
14 $\hat{P}_p \leftarrow \hat{P}_c; \hat{P}_c \leftarrow \hat{P}_c$.next
15 **end while**
16 **Accept** j_k.
17 UPDATE$(\hat{P}_h, \delta_x, \delta_y)$
18 $\bar{d}_{last} \leftarrow \bar{d}_k$

UPDATE($\hat{P}_h, \delta_x, \delta_y$)

```
1  P̂_p ← P̂_h
2  SHIFT(P̂_p, δ_x, δ_y)
3  P̂_c ← P̂_p.next
4  while P̂_c not null
5          SHIFT(P̂_c, δ_x, δ_y)
6          j ← ⌈log_{1+ε}(P̂_p.y_l)⌉; k ← ⌈log_{1+ε}(P̂_c.y_r)⌉
           ▷ P̂_p and P̂_c have moved to same region
7          if j == k
8                  MERGE(P̂_p, P̂_c)
9          P̂_p ← P̂_c; P̂_c ← P̂_c.next
10 end while
```

INSERT(x, y)

```
1  Allocate a new node P̂
2  P̂.x_l ← x; P̂.x_r ← x
3  P̂.y_l ← y; P̂.y_r ← y
4  P̂.next ← null
5  return P̂
```

DELETE($\hat{P}$)

```
1  Free the memory for node P̂.
```

SHIFT($\hat{P}, x, y$)

```
1  P̂.x_l ← P̂.x_l + x; P̂.x_r ← P̂.x_r + x
2  P̂.y_l ← P̂.y_l + y; P̂.y_r ← P̂.y_r + y
```

MERGE($\hat{P}_1, \hat{P}_2$)

```
1  if P̂_1.x_l > P̂_2.x_l
           ▷ Update left anchor
2          P̂_1.x_l ← P̂_2.x_l; P̂_1.y_l ← P̂_2.y_l
3  if P̂_1.y_r < P̂_2.y_r
           ▷ Update right anchor
4          P̂_1.x_r ← P̂_2.x_r; P̂_1.y_r ← P̂_2.y_r
5  P̂_1.next ← P̂_2.next;
6  DELETE(P̂_2)
```

Algorithm Correctness

We now show that our approximate admission controller is correct in the following theorem. Throughout this section, we consider a new element of $\mathbb{L}$ as an "approximation point" only after the admission controller APPROXIMATEAC(Λ, j_k, ϵ) has returned from its execution.

Theorem 13 *If* APPROXIMATEAC(Λ, j_k, ϵ) *returns "Accept", then* j_k *may be admitted without violating the demand-curve interface* Λ*.*

Our first lemma of this subsection shows that we always "cover" any deleted point by another point; in other words, any deleted point must be contained in the redundancy region of a point that is in list $\mathbb{L}$. Complete proof of all the lemmas in this section can be found in Appendix A.

Lemma 45 *For any approximation point* $\hat{P}$ *that was inserted into list* $\mathbb{L}$*, if* $\hat{P}$ *is deleted, then there exists some approximation point* $\hat{P}'$ *in* $\mathbb{L}$ *such that* $\hat{P}'.x_l \leq \hat{P}.x_l$ *and* $\hat{P}'.y_r \geq \hat{P}.y_r$*.*

The next lemma shows the correspondence between points stored in $\mathbb{L}$ and intervals ending at the deadline of last admitted job (i.e., $\bar{d}_{last}$).

Lemma 46 *After the call to* APPROXIMATEAC-INIT() *and the* k*'th invocation of* APPROXIMATEAC(Λ, j_k, ϵ) *where* $k \in \mathbb{N}$*, for* MAD*-sequenced jobs* $j_1, j_2, \ldots, j_k$*, define* J_k *to be the set of jobs admitted by the approximate admission controller. For each job* $j_\ell \in J_k$*, there exists an approximation point* $\hat{P}$ *in list* $\mathbb{L}$ *such that* $\hat{P}.x_l$ *is at most* $\bar{d}_{last} - A_\ell$ *and* $\hat{P}.y_r$ *is at least* $\mathsf{demand}(J_k, A_\ell, \bar{d}_{last})$*. Furthermore,* $\bar{d}_{last}$ *equals* $\max\big\{\{0\} \cup \{\bar{d}_\ell \mid j_\ell \in J_k\}\big\}$.

Proof: We prove the lemma by induction on k.

Base Case: When $k = 0$, APPROXIMATEAC-INIT() has been invoked and thus no jobs have been admitted; i.e, $J_0 = \emptyset$. The lemma is clearly true as $\mathbb{L}$ is initialized to $\emptyset$ and $\bar{d}_{last}$ is initialized to zero.

Inductive Hypothesis: Assume that the lemma holds for each i ($i = 1, 2, \ldots, k-1$) successive calls to APPROXIMATEAC(Λ, j_i, ϵ).

Inductive Step: We must show that the lemma holds for the k'th call to APPROXIMATEAC(Λ, j_k, ϵ). The admission controller can either return "accept" or "reject". Let us first consider the case that APPROXIMATEAC(Λ, j_k, ϵ) returns "reject". Then, J_{k-1} is identical to J_k and $\bar{d}_{last}$ is not changed by any instruc-

tion in the execution path to "reject". Thus, by the inductive hypothesis, the lemma obviously continues to hold as the state is identical to after the call to APPROXIMATEAC(Λ, j_k, ϵ).

Now, consider the case when APPROXIMATEAC(Λ, j_k, ϵ) returns "accept". Line 18 of the procedure sets $\bar{d}_{last}$ equal to $\bar{d}_k$; Let the updated value of $\bar{d}_{last}$ and $\mathbb{L}$ be denoted by $\bar{d}_{last}^{\text{new}}$ and $\mathbb{L}^{\text{new}}$ respectively. Let $\bar{d}_{last}^{\text{old}}$ and $\mathbb{L}^{\text{old}}$ denote the value of the $\bar{d}_{last}$ and $\mathbb{L}$, prior to the invoke of APPROXIMATEAC(Λ, j_k, ϵ). By the inductive hypothesis, for each job $j_\ell \in J_{k-1}$, there exists $\hat{P} \in \mathbb{L}^{\text{old}}$ such that $\hat{P}.x_l$ is at most $\bar{d}_{last}^{\text{old}} - A_\ell$ and $\hat{P}.y_r$ is at least $\mathsf{demand}(J_{k-1}, A_\ell, \bar{d}_{last}^{\text{old}})$. The SHIFT subroutine shifts each approximation point (and its anchors) to the right by $\delta_x = \bar{d}_{last}^{\text{new}} - \bar{d}_{last}^{\text{old}}$ and up by $\delta_y = E_k$. Lemma 45, implies that if any approximation point $\hat{P}$ is deleted by a MERGE, then $\mathbb{L}^{\text{new}}$ has an approximation point $\hat{P}'$ with $\hat{P}'.x_l \leq \hat{P}.x_l$ and $\hat{P}.y_r \leq \hat{P}'.y_r$. Thus, each $\hat{P} \in \mathbb{L}^{\text{old}}$ is now $(\hat{P}.x_l + \delta_x, \hat{P}.y_r + \delta_y) \in \mathbb{L}^{\text{new}}$ or has an approximation point $\hat{P}'$ that covers it. Furthermore, $\hat{P}.x_l + \delta_x$ is at most $(\bar{d}_{last}^{\text{new}} - \bar{d}_{last}^{\text{old}}) + \bar{d}_{last}^{\text{old}} - A_\ell = \bar{d}_{last}^{\text{new}} - A_\ell$ and $\hat{P}.y_r + \delta_y$ is at most $\mathsf{demand}(J_{k-1}, A_\ell, \bar{d}_{last}^{\text{old}}) + E_\ell$. The last expression is equivalent to $\mathsf{demand}(J_k, A_\ell, \bar{d}_{last}^{\text{new}})$ since increasing the interval length by δ_x includes only the new job j_k in the interval $[A_\ell, \bar{d}_{last}^{\text{new}}]$. Finally, adding the point $\{(D_k - \delta_x, E_k - \delta_y)\}$ in Line 4 and shifting by δ_x and δ_y is equivalent to adding an approximation point $\hat{P} = (D_k, E_k)$ which equals $(\bar{d}_{last}^{\text{new}} - A_k, \mathsf{demand}(J_k, A_k, \bar{d}_{last}^{\text{new}}))$ to the list $\mathbb{L}$. Thus, the lemma holds for J_k. ■

The next lemma shows that the approximate admission controller returns "Accept" for a job j_k only when it is safe to do so.

Lemma 47 *After the call to* APPROXIMATEAC-INIT() *and the k'th invocation of* APPROXIMATEAC(Λ, j_k, ϵ) *where $k \in \mathbb{R}$, for* MAD*-sequenced jobs $j_1, j_2, \ldots, j_k$, define J_k to be the set of jobs admitted by the approximate admission controller. It must be that Equation 6.1 holds for J_k and Λ.*

Theorem 13 immediately follows from the above lemma.

Approximation Ratio

In this section, we argue that when the approximate admission controller rejects a job, then the exact admission controller would also have done so on a slightly "smaller" dbi. The accuracy of the test is determined by the accuracy parameter $\epsilon > 0$. Note that, in online setting, set of jobs accepted by the approximate algorithm might not be a subset of set of jobs accepted by the exact algorithm. However, for the ease of analysis, in this section we assume that initially both the algorithms have same set of

accepted jobs. While not required for the correctness of the algorithm, to prove the approximation ratio, we do require that each job j_i have execution time E_i at least equal to one. (The algorithm will still work correctly for $E_i < 1$; however, the approximation ratio is not true in this case).

Our first lemma of the subsection shows that the ratio of y-values of the right and left anchor point is bounded in terms of ϵ.

Lemma 48 *For any approximation points $\hat{P}$ the following invariant holds:*

$$\hat{P}.y_r \leq (1+\epsilon)\hat{P}.y_l. \tag{6.11}$$

The following corollary follows from the observation that for any approximation point $\hat{P}$ a left anchor point below $\mathcal{A}^{i-1}.lb$ and a right anchor point above $\mathcal{A}^{i-1}.ub$ (equal to $\mathcal{A}^{i}.lb$) would violate the invariant of Lemma 48.

Corollary 14 *For an approximation point $\hat{P}$ with $\mathcal{A}^i.lb \leq \hat{P}.y_r \leq \mathcal{A}^i.ub$ and $\hat{P}.y_l < \mathcal{A}^i.lb$ where $i > 1$, the left anchor point of $\hat{P}$ must be in the $(1+\epsilon)$-region $\mathcal{A}^{i-1}$; that is, $\mathcal{A}^{i-1}.lb \leq \hat{P}.y_l \leq \mathcal{A}^{i-1}.ub$.*

The next lemma shows that for any approximation point, the left anchor corresponds to the exact demand over some interval.

Lemma 49 *After the call to* APPROXIMATEAC-INIT() *and the k'th invocation of* APPROXIMATEAC(Λ, j_k, ϵ) *where $k \in \mathbb{N}$, for* MAD*-sequenced jobs $j_1, j_2, \ldots, j_k$, define J_k to be the set of jobs admitted by the approximate admission controller. For each $\hat{P}$ in $\mathbb{L}$, it must be that there exists a $j_\ell \in J_k$ such that* $\mathsf{demand}(J, A_\ell, \bar{d}_{last})$ *equals $\hat{P}.y_l$ and $\bar{d}_{last} - A_\ell$ equals $\hat{P}.x_l$.*

We may now quantify the inaccuracy of our approximate admission controller by proving the following theorem.

Theorem 14 *Given a set of previously-admitted jobs J, if* APPROXIMATEAC(Λ, j_k, ϵ) *returns "Reject", then* EXACTAC(Λ, j_k) *also returns "Reject" for a demand-curve $\frac{1}{1+\epsilon}\mathsf{dbi}(\Lambda, \cdot)$ on the same previously-admitted job set.*

Proof: If APPROXIMATEAC(Λ, j_k, ϵ) returns "Reject" given previously admitted job set J, then there exists an approximation point $\hat{P}$ that fails the test in either Line 1 (i.e., the execution of j_k is too large over

it's arrival to deadline interval), Line 8 (i.e., it would fail after the SHIFT operation is applied), or Line 12 (i.e., it would fail after being merged with another point in the same $(1+\epsilon)$-region). Let us first assume that $\hat{P}$ would fail the test of Line 1; clearly j_k would fail the test of Line 4 of EXACTAC.

By Lemma 49, $\hat{P}.x_l$ equals $\bar{d}_{last} - A_\ell$ and $\hat{P}.y_l$ equals $\mathsf{demand}(J, A_\ell, \bar{d}_{last})$ for some $J_\ell \in J$. If $\hat{P}$ fails in Line 8, then $\mathsf{dbi}(\Lambda, \hat{P}.x_l + \delta_x) < \hat{P}.y_r + \delta_y$ which implies that $\mathsf{dbi}(\Lambda, \bar{d}_k - A_\ell) < \hat{P}.y_r + \delta_y \leq (1+\epsilon)\hat{P}.y_l + \delta_y = (1+\epsilon)\mathsf{demand}(J, A_\ell, \bar{d}_{last}) + E_k \leq (1+\epsilon)\mathsf{demand}(J \cup \{j_k\}, A_\ell, \bar{d}_k)$ by Lemmas 48 and 49. We may similarly show that the lemma holds if Line 12 fails. Thus, the approximation ratio holds in all cases. ■

Algorithm Complexity

The complexity of the admission controller depends on the size of the linked list, since both APPROXIMATEAC and UPDATE have a loop traversing the list from beginning to end. We argue that the number of approximation points (nodes) for each $(1+\epsilon)$-region is constant (at most two) in Theorem 15. Therefore, the complexity of the algorithm is directly proportional to the number of $(1+\epsilon)$-regions.

Theorem 15 *For $\forall i$, a $(1+\epsilon)$-region $\mathcal{A}^i$ contains at most two approximation points.*

Our first lemma to prove the above theorem shows that each newly-admitted job corresponds to a node at the front of list $\mathbb{L}$.

Lemma 50 *After the call to* APPROXIMATEAC-INIT() *and the k'th invocation of* APPROXIMATEAC(Λ, j_k, ϵ) *where $k \in \mathbb{N}$ and $\epsilon > 0$, for* MAD*-sequenced jobs $j_1, j_2, \ldots, j_k$, define J_k to be the set of jobs admitted by the approximate admission controller. The approximation point $\hat{P} = (D_k, E_k)$ is the first node of list $\mathbb{L}$; i.e., the newly-admitted job is at the front of the list.*

Based on this observation, the following lemma states that for any two approximation point $\hat{P}_1$ and $\hat{P}_2$ the vertical region between the anchor points of the two points does not overlap. In other words, the anchor points are totally ordered with respect to their y-values.

Lemma 51 *For any two approximation points $\hat{P}_1$ and $\hat{P}_2$ in list $\mathbb{L}$ where $\hat{P}_1$ precedes $\hat{P}_2$, the following invariant holds:*

$$\hat{P}_1.y_l \leq \hat{P}_1.y_r \leq \hat{P}_2.y_l \leq \hat{P}_2.y_r. \tag{6.12}$$

Using this lemma, we will prove that the list $\mathbb{L}$ will have at most two nodes corresponding to a $(1+\epsilon)$-region.

Lemma 52 *For $\forall i$, $(1+\epsilon)$-region $\mathcal{A}^i$ contains at most one approximation point such that $(\mathcal{A}^i.lb \leq \hat{P}.y_r \leq \mathcal{A}^i.ub) \wedge (\mathcal{A}^i.lb \leq \hat{P}.y_l \leq \mathcal{A}^i.ub)$ after each call to* APPROXIMATEAC.

The next corollary shows that at most one approximation point may "straddle" any $(1+\epsilon)$-region boundary. In other words, at most one approximation point has left anchor point below and right anchor point above the boundary $\mathcal{A}^i.lb$ for any $i > 1$. This corollary follows by the observation that if there were two such approximation points, then they would have to overlap in terms of y-value which would contradict the invariant of Equation 6.12 of Lemma 51.

Corollary 15 *For $\forall i$, region $\mathcal{A}^i$ contains at most one approximation point such that $(\mathcal{A}^i.lb \leq \hat{P}.y_r \leq \mathcal{A}^i.ub) \wedge (\hat{P}.y_l \leq \mathcal{A}^i.lb)$.*

Combining Lemma 52 and Corollary 15, we see that at most one approximation point exist completely inside any region $\mathcal{A}^i$ and at most one straddles the boundary of $\mathcal{A}^i.lb$ for $i \geq 1$. This proves Theorem 15, which upper bounds the number of intervals that the approximate admission controller tracks.

Despite bounding the number of approximation points per $(1+\epsilon)$-region, the number of these regions could be potentially infinite, for an arbitrary dbi; however, practically, a system cannot define an arbitrarily infinite dbi function. Typically, there are two design choices: 1) store the dbi as a finite set of linear segments (i.e., the dbi has a finite number of steps and an entry for each step in the function); or 2) dbi is generated from some finite set of recurring tasks and each point can be calculated using some known closed-form equation. In the next section, we explore the second option by showing how to modify the algorithm to handle a dbi generated from periodic/sporadic task systems. Thus, for now, we will assume that we are given a finite number of line segments as dbi.

Let r be the number of line segments required to specify the dbi for Λ. We can view the dbi as an ordered set $\{((a_i, b_i), s_i) \mid 1 \leq i \leq r\}$ where elements are ordered in increasing a_i values and (a_i, b_i) represents the left endpoint of the i'th line segment and s_i is the slope of the line segment. In other words, for any interval length $t \in [a_i, a_{i+1})$ for some $i : 1 \leq i < r$ the $\mathsf{dbi}(\Lambda, t) = b_i + (t - a_i)s_i$. Furthermore, to ensure that dbi is non-decreasing, $b_i + (a_{i+1} - a_i)s_i \leq b_{i+1}$ for all $i : 1 \leq i < r$. For approximation points $\hat{P}.x_l$ larger than a_r, we may modify APPROXIMATEAC in Algorithm 9 to use a

constant-time approach given in Section 6.2 (Algorithm 6) for doing admission control of a single-step dbi by storing only a single point with $\hat{P}.x_l \geq a_r$. Otherwise, let $\mathcal{U}$ equal $b_{r-1} + (a_r - a_{r-1})s_{r-1}$. For all points before a_r, APPROXIMATEAC requires at most $\lceil \log_{1+\epsilon} \mathcal{U} \rceil$ different $(1+\epsilon)$-regions to cover all the approximation points. Thus, we have at most $O(\log_{1+\epsilon} \mathcal{U})$ approximation points. To calculate the dbi at any of the approximation points, we must simply look up the value in the ordered list which may be accomplished in $O(\log r)$ time complexity. Thus, the overall complexity of the approach for a finite stored dbi is $O((\log_{1+\epsilon} \mathcal{U})(\log r))$. The value of $\mathcal{U}$ is at most exponential (with base of two) in the number of bits to represent the line segments of the dbi. Our approximate admission controller therefore has complexity that is polynomial in the number of bits to store each line segment and $1/\epsilon$. Furthermore, the worst-case computational complexity does not depend on the number of jobs admitted in during the lifetime of the system; this removes a fundamental drawback of the exact admission controller.

We derive an upper bound on periodic dbi in Section 6.4. Let the upper bound for the dbi is $\mathcal{U}$, then the number of regions will be $\log_{1+\epsilon} \mathcal{U}$ by Definition 17. Since all other operations take constant time, the complexity of the algorithm is $O(\log_{1+\epsilon} \mathcal{U})$.

6.4 Deriving Upper-bound for dbi

In this section, we consider the second possibility posed in Section 6.3.3: the demand-curve interface is a periodic function. In particular, we assume that we are given a demand-curve generated from n constrained-deadline sporadic tasks [70]: $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$ (Section 3.1.2). Each task τ_i is specified by the tuple (e_i, d_i, p_i) where e_i is the worst-case execution requirement, d_i is the relative deadline, and p_i is the period of the task; we further assume that $d_i \leq p_i$ for all τ_i. U_τ denotes the total utilization of τ, and equal to $\sum_{i=1}^{n} e_i/p_i$. It has been shown [20] that the demand-curve $\mathsf{dbi}(\tau, t)$ equals $\sum_{\tau_i \in \tau} \lfloor (t - d_i)/p_i + 1 \rfloor \cdot e_i$ (Equation 3.2). Furthermore, it can be easily shown that this dbi repeats with hyperperiod H_τ equal to $\mathsf{lcm}_{i=1}^{n}\{p_i\}$ (Section 3.2.2); this repetition implies

$$\forall 0 \leq t \leq H_\tau :: \mathsf{dbi}(\tau, t + H_\tau) = \mathsf{dbi}(\tau, t) + U_\tau H_\tau. \tag{6.13}$$

Figure 6.5 shows the periodic behavior of the dbi. Note in the figure, we have divided the xy-plane into rectangular regions labeled with roman numerals. The width of each region is H_τ and the height

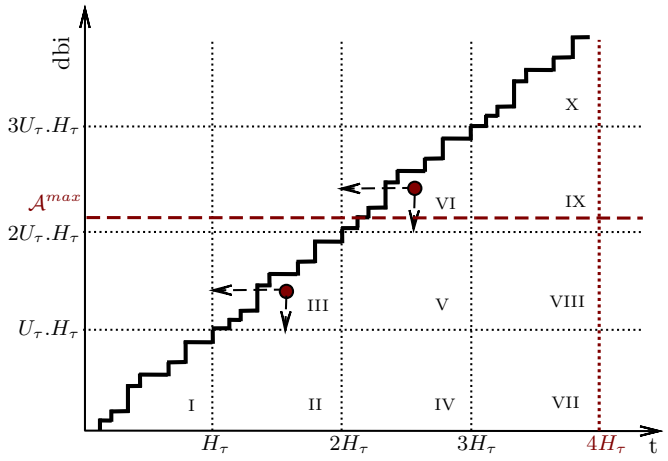


Figure 6.5: Deriving Upper Bound for Periodic dbi.

is $U_\tau H_\tau$. Due to the periodic nature of the dbi, Regions I, III, VI, X, etc. are identical; similarly, the Regions II, V, IX, etc. are also identical. Clearly a point (x, y) in any region with $x \geq H_\tau$ and $y \geq U_\tau H_\tau$ can be "mapped" to a point in its identical region (i.e., one region down and one region to the left) by $(x - H_\tau, y - U_\tau H_\tau)$. From Equation 6.13, we may observe that

$$\mathsf{dbi}(\tau, x - H_\tau) \geq y - U_\tau H_\tau \Leftrightarrow \mathsf{dbi}(\tau, x) \geq y. \tag{6.14}$$

Given the above observations regarding $\mathsf{dbi}(\tau, t)$, we now present a modification of the approximate admission controller to limit the number of regions to polynomial in ϵ and the size of τ. Let $\mathcal{A}^{\max}$ be the first $(1+\epsilon)$-region completely above $2U_\tau H_\tau$; the region $\mathcal{A}^{\max}$ represents the last region that we will track. When an approximation point $\hat{P}$ crosses $\mathcal{A}^{\max}.lb$ (i.e., $\hat{P}.y_l$ crosses this boundary), we check whether the point entered via Region VI or IX; if so, we use the mapping of Equation 6.14 to map the point back to Region III or V, respectively. Otherwise, if it crossed $\mathcal{A}^{\max}.lb$ in some other region, we may show that if this point ever violates the dbi, there exists a smaller interval that also does. Therefore, we can remove this point from consideration. We test for this by checking the condition $\hat{P}.x_l > 4H_\tau$. Algorithm 10 shows this procedure called dbi- WRAPCHECK which we may add in the UPDATE subroutine in APPROXIMATEAC after the SHIFT of each node. We can show that the number of $(1 + \epsilon)$-regions for a periodic dbi is $O(\log_{1+\epsilon} U_\tau H_\tau)$ based the definition of $\mathcal{A}^{\max}$. Since $u \leq n$ and $H_\tau \leq p_{\max}^n$ where $p_{max} = \max_{\tau_i}\{p_i\}$, then the total number of regions can be upper-bounded by $O(n \log_{1+\epsilon} p_{\max})$. The time to compute the periodic dbi for any interval length adds an additional factor of $O(n)$ to the time complexity.

Algorithm 10 Pseudo-code for checking upper bound of periodic dbi for admission control of MAD jobs.

dbi-WRAPCHECK($\Lambda, \hat{P}, \epsilon$)

```
1  ▷ Let A^max be the first region above 2U_τ H_τ.
2  if P̂.x_l > 4H_τ
3        Delete all the nodes from P̂ to the end of the list.
4  if P̂.y_l > A^max.lb
5        P̂.x_l ← P̂.x_l − H_τ
6        P̂.y_r ← P̂.y_r − U_τ H_τ
7        r ← ⌈log_{1+ε}(P̂.y_l)⌉
8        if A^r has approximation point (P̂_1)
9              MERGE(P̂_1, P̂)
10       else
11             INSERT(x_l, y_r)
```

6.4.1 Eliminating Approximation Points for Periodic dbi

In this section, we show that we may discard an approximation point $\hat{P}$ if it is "too far away" from $\mathsf{dbi}(\Lambda, \cdot)$ for a dbi generated from a set of sporadic tasks τ. The first lemma gives a technical result regarding periodic dbis.

Lemma 53 *For any positive real numbers x and y, it must be that*

$$\mathsf{dbi}(\Lambda, x+y) - \mathsf{dbi}(\Lambda, y) \geq \mathsf{dbi}(\Lambda, x). \tag{6.15}$$

Proof: We use the following property of floors that for any a and b: $\lfloor a \rfloor + \lfloor b \rfloor \leq \lfloor a+b \rfloor$. By $d_i \leq p_i$ and the definition of dbi for sporadic tasks, we have

$$\begin{array}{l} \sum_{i=1}^{n} \left(\left\lfloor \frac{x+y-d_i}{p_i} \right\rfloor + 1 \right) e_i - \sum_{i=1}^{n} \left(\left\lfloor \frac{y-d_i}{p_i} \right\rfloor + 1 \right) e_i \\ = \sum_{i=1}^{n} \left(\left\lfloor \frac{x+y-d_i}{p_i} \right\rfloor - \left\lfloor \frac{y-d_i}{p_i} \right\rfloor \right) e_i. \end{array} \tag{6.16}$$

Letting $a = \frac{x}{p_i}$ and $b = \frac{y-d_i}{p_i}$ and applying the property of floors, implies the lemma. ■

The next lemma shows that if the left anchor point of an approximation point $\hat{P}$ passes the boundary of $4H_\tau$ and its y-value less than $2U_\tau H_\tau$, then there exists another point (with smaller x-value) that will "cover" $\hat{P}$.

Lemma 54 *Let $\hat{P}$ be an approximation point such that $\hat{P}.x_l > 4H_\tau$ and $\hat{P}.y_r \leq 2U_\tau H_\tau$ at some time $T > 0$ and for any sequence of admitted jobs after T, $\hat{P}$ is not merged with any other point. Let $\hat{P}^{\text{new}}$ be the approximation point corresponding to $\hat{P}$ after jobs arriving between T and T' have been admitted where $T < T'$. If $\hat{P}^{\text{new}}.y_r > \mathsf{dbi}(\Lambda, \hat{P}^{\text{new}}.x_l)$ at time T', then there exists $\hat{P}' \in L$ at time T' such that $\hat{P}'.x_l \leq \hat{P}.x_l$ and $\hat{P}'.y_r > \mathsf{dbi}(\Lambda, \hat{P}'.x_l)$.*

Proof: Let J be the set of jobs admitted by APPROXIMATEAC before T. Let J' be the set of jobs admitted after T up until T'. Observe by supposition of the lemma, $\hat{P}.x_l > 4H_\tau$ which implies that $\mathsf{dbi}(\Lambda, \hat{P}.x_l) > \mathsf{dbi}(\Lambda, 4H_\tau) = 4U_\tau H_\tau$. Since $\hat{P}.y_r \leq 2U_\tau H_\tau$, it must be that

$$\mathsf{dbi}(\Lambda, \hat{P}.x_l) - \hat{P}.y_r \geq 2U_\tau H_\tau. \tag{6.17}$$

Since $\hat{P}$ is not merged, only SHIFT operations will change the location of $\hat{P}$. Let $J = \{j_1, j_2, \ldots, j_k\}$ and $J' = \{j_{k+1}, \ldots, j_n\}$. We denote the value of the variable $\bar{d}_{last}$ at time T as $\bar{d}^{\text{old}}_{last}$; similarly, $\bar{d}^{\text{new}}_{last}$ represents the variable at time T'. Please note by our definition of J and J', it must be that $\bar{d}^{\text{old}}_{last} = \bar{d}_k$ and $\bar{d}^{\text{new}}_{last} = \bar{d}_n$. By Lemma 46, there exists $j_i \in J$ such that $\hat{P}.x_l \leq \bar{d}^{\text{old}}_{last} - A_i = \bar{d}_k - A_i$ and $\hat{P}.y_r \geq \mathsf{demand}(J, A_i, \bar{d}_k)$. Since j_k was accepted $\hat{P}.y_r \leq \mathsf{dbi}(\Lambda, \hat{P}.x_l) \leq \mathsf{dbi}(\Lambda, \bar{d}_k - A_i)$.

Let $\bar{\delta}_x$ equal $\bar{d}^{\text{new}}_{last} - \bar{d}^{\text{old}}_{last}$. Let $\bar{\delta}_y$ equals the total execution of all jobs of J'. The approximation point $\hat{P}^{\text{new}}$ is the original approximation point $\hat{P}$ shifted to the right by $\bar{\delta}_x$ and upwards by $\bar{\delta}_y$. If $\hat{P}^{\text{new}}.y_r > \mathsf{dbi}(\Lambda, \hat{P}^{\text{new}}.x_l)$ is true, then

$$\hat{P}.y_r + \bar{\delta}_y > \mathsf{dbi}(\Lambda, \hat{P}.x_l + \bar{\delta}_x). \tag{6.18}$$

Combining Equations 6.17 and 6.18, we obtain

$$\bar{\delta}_y > \mathsf{dbi}(\Lambda, \hat{P}.x_l + \bar{\delta}_x) - \mathsf{dbi}(\Lambda, \hat{P}.x_l) + 2U_\tau H_\tau. \tag{6.19}$$

Now consider the interval $[A_{k+1}, \bar{d}_n]$. Observe that $\bar{d}_n - A_i = (\bar{d}_n - A_{k+1}) + (A_{k+1} - A_i)$. By Lemma 53,

$$
\begin{aligned}
&\mathsf{dbi}(\Lambda, \bar{d}_n - A_{k+1}) \\
&\quad \leq \mathsf{dbi}(\Lambda, \bar{d}_n - A_i) - \mathsf{dbi}(\Lambda, A_{k+1} - A_i) \\
&\quad \leq \mathsf{dbi}(\Lambda, \hat{P}.x_l + \bar{\delta}_x) - \mathsf{dbi}(\Lambda, A_{k+1} - A_i) \\
&\quad \leq \mathsf{dbi}(\Lambda, \hat{P}.x_l + \bar{\delta}_x) - \Big[\mathsf{dbi}(\Lambda, \hat{P}.x_l) - U_\tau H_\tau\Big]
\end{aligned}
\tag{6.20}
$$

The last inequality follows since $\bar{d}_{k+1} \geq \bar{d}_k$ (due to the MAD property). Assuming that $D_{k+1} \leq H_\tau$, $\mathsf{dbi}(\Lambda, A_{k+1} - A_i) \geq \mathsf{dbi}(\Lambda, (\bar{d}_k - H_\tau) - A_i) = \mathsf{dbi}(\Lambda, \bar{d}_k - A_i) - U_\tau H_\tau \geq \mathsf{dbi}(\Lambda, \hat{P}.x_l) - U_\tau H_\tau$.

By Lemmas 45 and 46, there exist an approximation point $\hat{P}' \in \mathbb{L}$ at time T' such that $\hat{P}'.x_l \leq \bar{d}_n - A_{k+1}$ and $\hat{P}'.y_r \geq \mathsf{demand}(J \cup J', A_{k+1}, \bar{d}_n) = \bar{\delta}_y$. Combining these facts with Equations 6.17 and 6.20, we obtain

$$\hat{P}'.y_r > \mathsf{dbi}(\Lambda, \hat{P}.x_l + \bar{\delta}_x) - \mathsf{dbi}(\Lambda, \hat{P}.x_l) + 2U_\tau H_\tau \tag{6.21}$$

and

$$\mathsf{dbi}(\Lambda, \hat{P}'.x_l) \leq \mathsf{dbi}(\Lambda, \hat{P}.x_l + \bar{\delta}_x) - \mathsf{dbi}(\Lambda, \hat{P}.x_l) + U_\tau H_\tau. \tag{6.22}$$

Clearly, Equations 6.21 and 6.22 implies that $\hat{P}'.y_r > \mathsf{dbi}(\Lambda, \hat{P}'.x_l)$; if $\hat{P}'.x_l \leq 4H_\tau$, then the lemma is proved. If not, we can repeat the same logic above to find $\hat{P}''$ with $\hat{P}''.y_r > \mathsf{dbi}(\Lambda, \hat{P}''.x_l)$ where $\hat{P}''.x_l < \hat{P}'.x_l$. ■

6.4.2 Server Design

Our admission controllers ensure that the total system demand for the admitted jobs will never violate the demand-curve interface for the subsystem by policing the jobs before executing them. However, we need a mechanism to strictly enforce the interface at runtime; for example, if a job needs to execute more than its worst-case execution time, the system must ensure that temporal isolation is still maintained. We denote the worst-case execution time E_i specified in our aperiodic job model as the *estimated execution time* and $\bar{E}_i$ as the *actual execution time* of job j_i. In this section we address how the overrun ($E_i < \bar{E}_i$) and underrun ($E_i > \bar{E}_i$) situations can be handled so that the interface is not violated. We explore the design of a lightweight server to enforce temporal isolation and reclaim any unused reservation from a job that finishes early.

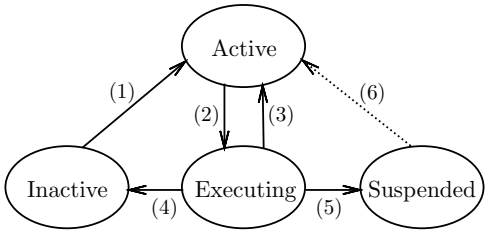


Figure 6.6: State transition diagram.

Temporal Isolation

Each job will have a server associated with it, which will monitor the execution time of the job. If the actual execution goes beyond the estimated execution time, the server will stop running the job. Upon j_i's admittance to the system (via our admission controller), the server is given a budget equal to E_i. The server consumes this budget at a linear rate only when executing the job. When the budget of the server for j_i is depleted, its execution is halted. If j_i has not successfully completed at this point, we have an overrun situation. The subsystem designer needs to determine whether to abort the job or spawn a new server by using the admission control algorithm with a new estimate of the remaining execution. Note that since the server does not permit the actual execution $\bar{E}_i$ of a job to exceed E_i, it enforces strong temporal isolation between subsystems.

Each server assigns a *state* to the admitted job. A job can be in one of the four states: *inactive*, *active*, *executing* and *suspended*. An inactive state is initial state of a job that just arrived to the system. Once the admission controller admits the job, it moves to active state (transition (1)). The job moves to executing state when the scheduler starts executing the job (transition (2)), and goes back to active state when a higher priority job preempts the execution of the job (transition (3)). While in executing state, if the total execution of the job exceeds E_i, the server transitions the job to suspended state (5). The job goes back to inactive state when it finishes execution (transition 4). From the suspended state, the job might move back to active state (6), however, it must get accepted through the admission controller with its new execution requirement and deadline.

Resource Reclamation

Now consider that $\bar{E}_i$ is less than E_i. Using APPROXIMATEAC directly, it would be difficult to modify our data structure to reclaim the reserved execution for j_i. (We would have to shift points backwards).

Thus, to accommodate early completion, we do a lazy update to the linked list $\mathbb{L}$. We maintain another list Q called the *active jobs list*. This list stores all the jobs whose deadline have not yet elapsed. Let the sum of demands of the jobs in the active job list is called *active demand*. We store the current value of active demand for Q in a variable q.

When a job j_i arrives to the system, the admission controller decides whether to admit the job based on an "inflated" execution time which is the sum of the active demand (q) and E_i. If admitted, a node is added to the list Q at the end in deadline order (recall that the jobs arrive in MAD order), and active demand q is increased by E_i; however, we defer the updating of nodes of list $\mathbb{L}$. Upon completion of a job j_i in Q, we reclaim execution in the active demand by increasing q by $E_i - \bar{E}_i$. When the deadline of any job j_i in the active job list elapses, it is removed from the front of Q and the active demand q is reduced by $\bar{E}_i$. Then, a new interval corresponding to $[A_i, \bar{d}_i]$ is inserted to $\mathbb{L}$, with the job's actual execution time $\bar{E}_i$ ($\leq E_i$), and update to existing intervals is performed with $(\delta_x, \delta_y) = (\bar{d}_i - \bar{d}_{last}, \bar{E}_i)$. In this way, it is ensured that if actual execution is less than the estimated, the remaining resource is reclaimed. Also, note that the size of Q is at most the number of active jobs N in the system.

Many admission controllers have the ability to "reset" upon a system idle point. Clearly, it is desirable to be able to reset the demand of a subsystem to zero at such a point. However, it is not possible to implement such a subsystem reset for arbitrary demand-curve interface model, where complete global knowledge of the state of all other subsystems comprising the system is unknown to a subsystem; doing so could result in a violation of the interface. Consider a subsystem $\mathcal{S}$ with interface $\mathsf{dbi}(\Lambda, t) = .9t$ and jobs $j_1(0, .9, 1)$ and $j_2(.91, .9, 1)$. If $\mathcal{S}$ executes all of j_1 immediately, then it will go idle at time .9. If $\mathcal{S}$ resets at this time, j_2 will be admitted even though j_1 and j_2 together clearly violate $\mathsf{dbi}(\Lambda, 1.91)$. In the next section we focus on determining reset points when the interface function has some specific characteristics.

6.5 Resetting the Admission Controller

While until now we have assumed that the dbi is an arbitrary function (i.e., piecewise continuous, non-negative, and non-decreasing function of interval lengths), which makes the determination of a subsystem reset point non-trivial. In this section we investigate few specific type of interface functions in which we will be able to reset the admission controller based on certain properties of the function. The basic idea is to determine a "safe" time interval $\mathcal{I}_{\mathcal{S}}$ for any subsystem $\mathcal{S}$ specified by an interface Λ such that if $\mathcal{S}$ is idle

for time interval $\mathcal{I}_\mathcal{S}$, we can safely reset $\mathcal{S}$ (i.e., discard all the demand points corresponding to previously admitted jobs in list $\mathbb{L}$). We assume that the system consists of MAD jobs.

Definition 20 (Subsystem reset time) *For any subsystem $\mathcal{S}$ specified with an interface Λ, and an arbitrary set of accepted* MAD *jobs J at any time T, the* subsystem reset time *$\mathcal{I}_\mathcal{S}$ represents the time interval for which if $\mathcal{S}$ is idle after the last admitted job j_{last}'s deadline $\bar{d}_{last}$, then it can forget its history of previously admitted jobs i.e., it can be reset safely.*

In the next section, we determine the subsystem reset time based on the characteristics of the interface function that specifies the subsystem. We derive subsystem reset point for *monotonic increasing-rate* dbi and *super-additive* dbi. Obtaining a general criteria for determining reset point of an arbitrary interface function remains as future work.

6.5.1 Monotonic Increasing-Rate dbi

A function is monotonic increasing-rate if the following property hold:

$$\frac{f(T_1)}{T_1} \leq \frac{f(T_1+T_2)}{T_1+T_2} \text{ where } T_1, T_2 \geq 0 \tag{6.23}$$

Few examples of monotonic increasing-rate functions are the uniprocessor supply function ($f(t) = t$), the supply-bound function for EDP or periodic resource model (Section 3.3.1), single-step demand interface (Section 3.3.2) etc. The monotonic increasing-rate property is due to the fact that the derivative of each of these functions are greater or equal zero. In the next lemma, we obtain the reset condition for subsystems specified by such interfaces.

Lemma 55 *Let subsystem $\mathcal{S}$ is specified by a monotonic increasing-rate interface Λ, that is, for $\forall T_1, T_2, \in \mathbb{R}$ and $T_1 \leq T_2$, $\mathsf{dbi}(\Lambda, T_1)/T_1 \leq \mathsf{dbi}(\Lambda, T_2)/T_2$. At time T, let J be the set of admitted jobs and $\bar{d}_{last}$ be the deadline of last admitted job j_{last}. The admission controller can be safely reset after any time point $T \geq \bar{d}_{last}$ when $\mathcal{S}$ is idle, that is, $\mathcal{I}_\mathcal{S} = 0$.*

Proof: Let $\mathcal{S}$ is idle at time $T \geq \bar{d}_{last}$, and a new job j_k arrives in the system with $A_k \geq \bar{d}_{last}$. To prove that the admission controller can be reset at time $T \geq \bar{d}_{last}$, we must show that if the demand over the

interval $[A_k, \bar{d}_k]$ does not violate dbi then the demand-points corresponding to already admitted set of jobs J will not violate dbi after adding j_k.

Since the admission controller already admitted the jobs in J, from Lemma 41 and 42 we obtain:

$$\forall j_i \in J, \mathsf{demand}(J, A_i, \bar{d}_{last}) \leq \mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i) \tag{6.24}$$

When j_k arrives, the EXACTAC(Λ, j_k) in Algorithm 8 checks at the very beginning (first iteration of for loop) that the new job j_k meets its demand E_k over the interval $[A_k, \bar{d}_k]$. That is:

$$\mathsf{demand}(j_k, A_k, \bar{d}_k) \leq \mathsf{dbi}(\Lambda, \bar{d}_k - A_k) \tag{6.25}$$

Given Equation 6.24 and 6.25, and a monotonic increasing rate dbi, we must show that when $A_k \geq \bar{d}_{last}$:

$$\forall i \leq k, \mathsf{demand}(J \cup j_k, A_i, \bar{d}_k) \leq \mathsf{dbi}(\Lambda, \bar{d}_k - A_i) \tag{6.26}$$

The worst-case demand for the new job is equal to the interface, i.e., $\mathsf{demand}(j_k, A_k, \bar{d}_k) = \mathsf{dbi}(\Lambda, \bar{d}_k - A_k)$. Since $\mathcal{S}$ is idle for the interval $[\bar{d}_{last}, A_k)$, this implies $\mathsf{demand}(j_k, \bar{d}_{last}, A_k) = 0$, thus, $\mathsf{demand}(j_k, A_k, \bar{d}_k) = \mathsf{demand}(j_k, \bar{d}_{last}, \bar{d}_k)$. Therefore, $\mathsf{demand}(j_k, \bar{d}_{last}, \bar{d}_k) = \mathsf{dbi}(\Lambda, \bar{d}_k - A_k) \leq \mathsf{dbi}(\Lambda, \bar{d}_k - \bar{d}_{last})$. Also we assume that for any of the prior demand-points stored by the algorithm for the interval $[A_i, \bar{d}_{last}], j_i \in J$, demand is maximum, that is, $\mathsf{demand}(J, A_i, \bar{d}_{last}) = \mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i)$. We have to show that the demand over the new interval $[A_i, \bar{d}_k]$ will meet the demand interface for this case.

By the monotonic increasing property of the dbi, we obtain:

$$\begin{aligned} & \frac{\mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i)}{\bar{d}_{last} - A_i} \leq \frac{\mathsf{dbi}(\Lambda, \bar{d}_k - A_i)}{\bar{d}_k - A_i} \\ \Rightarrow \quad & (\bar{d}_k - A_i)\mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i) \leq (\bar{d}_{last} - A_i)\mathsf{dbi}(\Lambda, \bar{d}_k - A_i) \end{aligned} \tag{6.27}$$

Similarly,

$$\begin{aligned} & \frac{\mathsf{dbi}(\Lambda, \bar{d}_k - \bar{d}_{last})}{\bar{d}_k - \bar{d}_{last}} \leq \frac{\mathsf{dbi}(\Lambda, \bar{d}_k - A_i)}{\bar{d}_k - A_i} \\ & \Rightarrow (\bar{d}_k - A_i)\mathsf{dbi}(\Lambda, \bar{d}_k - \bar{d}_{last}) \leq (\bar{d}_k - \bar{d}_{last})\mathsf{dbi}(\Lambda, \bar{d}_k - A_i) \end{aligned} \tag{6.28}$$

Table 6.1: Periodic dbi parameters

p	7	15	9	19	21	27	35	11
e	0.1	1.1	0.18	0.4	0.22	5.2	4	1.7

By adding Equation 6.27 and 6.29, we obtain:

$$\begin{aligned}
&(\bar{d}_k - A_i)(\mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i) + \mathsf{dbi}(\Lambda, \bar{d}_k - \bar{d}_{last})) \\
&\quad \leq (\bar{d}_k - \bar{d}_{last} + \bar{d}_{last} - A_i)\mathsf{dbi}(\Lambda, \bar{d}_k - A_i) \\
&=> \mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i) + \mathsf{dbi}(\Lambda, \bar{d}_k - \bar{d}_{last}) \leq \mathsf{dbi}(\Lambda, \bar{d}_k - A_i) \\
&=> \mathsf{dbi}(\Lambda, \bar{d}_{last} - A_i) + \mathsf{dbi}(\Lambda, \bar{d}_k - A_k) \leq \mathsf{dbi}(\Lambda, \bar{d}_k - A_i)
\end{aligned} \tag{6.29}$$

Similarly, for all other interval $[A_i, \bar{d}_k], j_i \in J$, it can be shown that the demand will be less or equal the dbi, which implies Equation 6.26. Thus, $\mathcal{S}$ can be safely reset once it is idle after last admitted job's deadline $\bar{d}_k$, hence, by Definition 20, reset time $\mathcal{I}_\mathcal{S}$ equals 0. ■

Corollary 16 *If a subsystem $\mathcal{S}$ is specified by a super-additive interface Λ, that is, for $\forall T_1, T_2 \in \mathbb{R}^+$, $\mathsf{dbi}(\Lambda, T_1) + \mathsf{dbi}(\Lambda, T_2) \leq \mathsf{dbi}(\Lambda, T_1 + T_2)$, then $\mathcal{S}$ can be reset at the instant it becomes idle after last admitted job's deadline, that is, $\mathcal{I}_\mathcal{S} = 0$.*

6.6 Simulation Results

In this section, we evaluate the performance of EXACTAC (Algorithm 8) and APPROXIMATEAC(Algorithm 9) by running them over synthetically generated MAD jobs. While we use the results obtained in Section 6.4 in the simulation, we do not include the results obtained in Section 6.5. During simulation we used following parameters and value ranges.

- We use a periodic demand interface with set of 8 periodic tasks with period p and execution e as shown in Table 6.1, and a total utilization of 0.5. The periods are randomly generated in the range $[5, 40]$ and task utilizations are generated using UUniFast algorithm [25]. The generated tasks have a hyperperiod H equal to 197505.

- For MAD job j_i, we generate following parameters from uniform distribution: interarrival time x between successive jobs is in the range $[0, 20]$ (i.e.,$A_i = A_{i-1} + x$); the relative deadline parameter

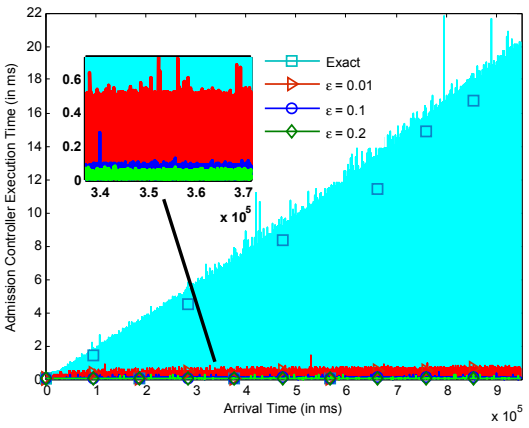


Figure 6.7: Execution Time vs Arrival Time of Jobs

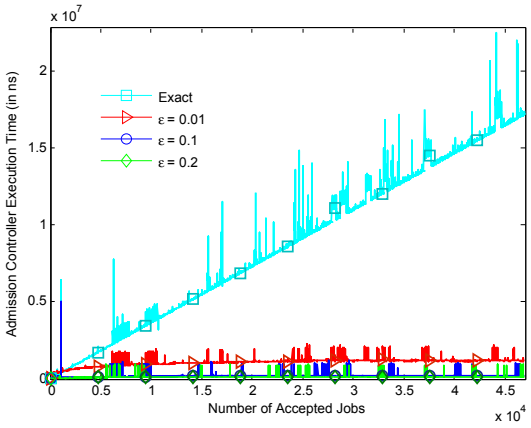


Figure 6.8: Execution Time vs Accepted Jobs

y is in the range $[0, 50]$ (i.e., $D_i = \max\{A_{i-1}, \bar{d}_{i-1}\} - A_i + y$); and the execution time E_i is in the range $[1, D_i]$.

- A 2.33 GHz Intel Core 2 Duo E6550 machine with 2.0GB RAM is used. The simulation runs until $A_i \geq 4H$.
- We use $\epsilon = [0.01, 0.1, 0.2]$.

We compare our proposed algorithms using two metrics: execution time and the number of accepted jobs over time. Figure 6.7 shows the execution time trace over time for each of the algorithms. Each point in the plot represents the execution time of corresponding algorithm in microseconds for the job arrival at time shown in the horizontal axis. Note, since this plot shows execution time for every run of the algorithm (it might accept or reject the job), the execution time highly fluctuates. Execution time is higher for the "accept" cases than "reject" cases, as it checks every interval in the list and updates all of them. On the other hand, an admission controller might "reject" a job at the very first line (see the algorithms in Section 6.3.1, 6.3.3), or after checking the corresponding list of intervals partially.

The next plot (Figure 6.8) compares the execution time required for acceptance (i.e., the worst-case) of the algorithms, given that the algorithm has already accepted the number of jobs indicated in the horizontal axis. For example, a point in the red curve at $x = 100$ represents time required by the approximate algorithm ($\epsilon = 0.01$) to admit a job in the system with already 100 jobs in the system. Note that set of accepted jobs for different algorithms will be different, however, the time required by the algorithm does not depend on job specific parameters, rather on the number of already accepted jobs by that algorithm. The

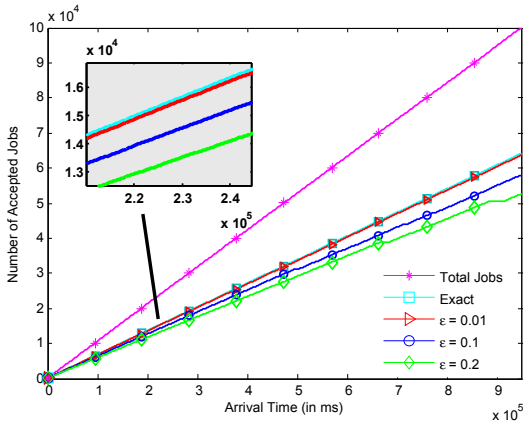


Figure 6.9: Accepted Jobs vs Arrival Time of Jobs

plot certifies this by showing linear growth in the execution time for the exact and saturated execution time for the approximate algorithms. Thus, we have a significant reduction in running time for the approximate admission controller over the infeasible exact admission controller.

The last plot shown in Figure 6.9 compares the number of jobs admitted by each of the algorithms over time. The leftmost curve represents total number of jobs that arrived in the system over time. The exact algorithm admits more jobs to the system then the approximate algorithms which is intuitive. We observe that the approximate algorithm with $\epsilon = 0.01$ performs very close to the exact algorithm.

6.7 Admission Controller for Arbitrary Aperiodic Jobs

In this section we relax the constraint of the MAD property for the aperiodic jobs in the system; that is, jobs may arrive in the system at any order of deadline (Figure 3.2 in Section 3.1.1). In this case, when a new job j_k is admitted to the system the demand over all intervals corresponding to the nodes in the list $\mathbb{L}$ (xy-plane data structure) will not increase the same amount (i.e., $\delta_y = E_k$). Since we insert nodes in $\mathbb{L}$ in deadline order, when a job arrives in the system in MAD order, we insert a new node at the beginning of the list corresponding to the new job's arrival to deadline interval, and update the existing nodes in the list to increase the intervals and to reflect the new job's demand in these intervals. This ensures that $\mathbb{L}$ is sorted in interval length order. When job j_k arrives in arbitrary order, then there might be some nodes (intervals) in $\mathbb{L}$ which represent intervals with arrival time later than j_k's arrival time A_k. Now, if these intervals correspond to jobs with absolute deadline earlier than j_k's absolute deadline, than the demand over these

intervals will not increase after admitting j_k. All the intervals which represent arrival time earlier than A_k and absolute deadline earlier than $\bar{d}_k$ will be updated similar to MAD case after admitting j_k in the system.

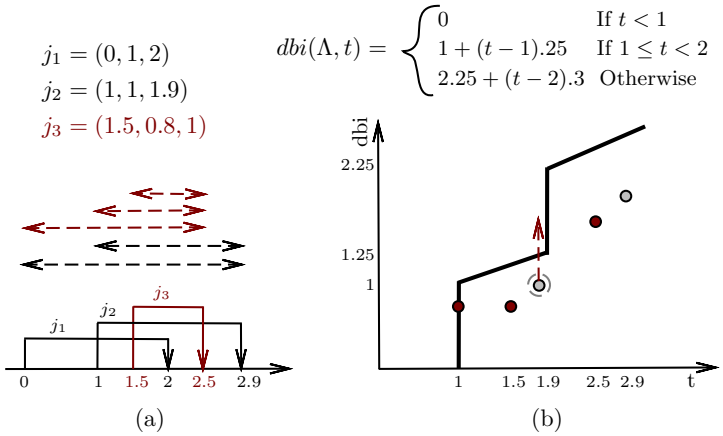


Figure 6.10: EXACTAC is insufficient for arbitrary aperiodic jobs.

First, we show via an example why the approach for MAD-sequenced job will not work for arbitrary aperiodic jobs. Consider our two-step dbi example (Section 6.3.2, Figure 6.2) and a sequence of aperiodic jobs $j_1(0, 1, 2), j_2(1, 1, 1.9)$, and $j_3(1.5, 0.8, 1)$ shown in Figure 6.10. For the $[A_i, \bar{d}_i]$ interval for each j_i, $\mathsf{dbi}(\Lambda, \bar{d}_i - A_i) > E_i$. The EXACTAC in Algorithm 8 will admit j_1. When j_2 arrives, the algorithm checks if the demand over the intervals $[A_i, D_2], i = 1, 2$ is met. When j_3 arrives (with $\bar{d}_3 < \bar{d}_2$), the algorithm will check whether all the intervals ending at $\bar{d}_3$ will meet their demand, i.e., the demand over the intervals $[A_i, \bar{d}_3], i = 1, 2, 3$ will not violate dbi (red points in Figure 6.10(b)). However, if j_3 is admitted the demand over the interval $[A_2, \bar{d}_2]$ will also increase by E_3 amount since $A_2 < A_3$ and $\bar{d}_2 > \bar{d}_3$, and this will cause a violation of the dbi, which EXACTAC will not be able to determine. Therefore, along with checking the demand of intervals ending at or before the new job's deadline, we need a procedure to check the demand over the intervals ending after the new job's deadline.

6.7.1 Algorithm Description

For each job arrival, the admission controller must ensure that the demand of all intervals representing both absolute deadline earlier than the new job or later than the new job are less or equal the dbi. We use a two step approach in this case to extend our admission controller for MAD jobs. We maintain a data structure B which contains admitted active jobs in the system ordered by their absolute deadline, similar to the list we used for admission control in single step demand interface in Section 6.2.2. Each node in B

Algorithm 11 Pseudo-code for approximate admission control of arbitrary aperiodic jobs where interface is an arbitrary demand-curve.

APPROXIMATEAC-A(Λ, j_k, ϵ)

```
1  if dbi(Λ, D_k) < E_k
2        Reject j_k.
3  Insert a node in B corresponding to j_k in absolute deadline order.
4  Copy the list 𝕃 to a list 𝕃'.
5  for each node in B corresponding to job j_i with d̄_i ≤ d̄_k
6        Perform UPONJOBDEADLINE-A(j_i) on list 𝕃', without
         deleting the node corresponding to j_i from B.
7        if any INSERT, SHIFT or MERGE violates dbi
8              Delete list 𝕃'. Remove node in B corresponding to j_k.
9              Reject j_k.
10 end for
11 Delete list 𝕃'.
12 Accept j_k.
```

UPONJOBDEADLINE-A(j_k)

```
1  δ_x ← d̄_k − d̄_last; δ_y ← E_k
2  UPDATE-A(δ_x, δ_y, j_k)
3  d̄_last ← d̄_k
4  Delete the node corresponding to j_k from B.
```

contains arrival time A_i, absolute deadline $\bar{d}_i$ and execution E_i for admitted active job j_i. If the admission controller accepts a job, it is first inserted into B and stored until its deadline elapse, at that time it is removed from B, and a node is added to $\mathbb{L}$ corresponding to this job, and existing nodes in $\mathbb{L}$ are updated to reflect the demand of this job.

A straightforward extension of APPROXIMATEAC in Algorithm 9 for MAD aperiodic jobs to approximate admission control of arbitrary aperiodic jobs is given in APPROXIMATEAC-A (Algorithm 11). We give the update operation in UPDATE-A, which inserts a new node to the list $\mathbb{L}$ in appropriate position (instead of at the head of the list for MAD case) and updates all the nodes accordingly. The procedure UPONJOBDEADLINE-A is invoked when the deadline of any node in list B elapses. Other operations on $\mathbb{L}$ for this algorithm (e.g., APPROXIMATEAC-INIT, INSERT, DELETE, SHIFT, MERGE) are similar to the subroutines shown in Algorithm 9. The steps for admission control procedure of an arbitrary aperiodic job are as follows:

UPDATE-A$(\delta_x, \delta_y, j_k)$

▷ Start from the head of the list
$\hat{P}_p \leftarrow$ null; $\hat{P}_c \leftarrow \hat{P}_h$
▷ Update intervals less than D_k
while $\hat{P}_c.x_r < D_k - \delta_x$
    SHIFT$(\hat{P}_c, \delta_x, 0)$
    $\hat{P}_p \leftarrow \hat{P}_c$; $\hat{P}_c \leftarrow \hat{P}_c$.next
end while
if $\hat{P}_c.x_l \leq D_k - \delta_x < \hat{P}_c.x_r$
    $\hat{P}_c.x_l \leftarrow \hat{P}_c.x_l + \delta_x$
    $\hat{P}_c.x_r \leftarrow \hat{P}_c.x_r + \delta_x$; $\hat{P}_c.y_r \leftarrow \hat{P}_c.y_r + \delta_y$
    if $\hat{P}_c.y_l$ and $\hat{P}_c.y_r$ not in same region
        Split the left and right anchors of $\hat{P}_c$ into two nodes $\hat{P}_c^l$ and $\hat{P}_c^r$
        Add $\hat{P} =$ INSERT$(D_k, \hat{P}_c^l.y_l + \delta_y)$, between $\hat{P}_c^l$ and $\hat{P}_c^r$
        if $\hat{P}_c^l$ and $\hat{P}$ are in same region
            MERGE$(\hat{P}_c^l, \hat{P})$; $\hat{P}_p = \hat{P}_c^r$
        else if $\hat{P}$ and $\hat{P}_c^r$ are in same region
            MERGE$(\hat{P}, \hat{P}_c^r)$; $\hat{P}_p = \hat{P}$
        else
            $\hat{P}_p = \hat{P}_c^r$
        $\hat{P}_c = \hat{P}_p$.next
    else
        $\hat{P}_p \leftarrow \hat{P}_c$; $\hat{P}_c \leftarrow \hat{P}_c$.next
else
    if $\hat{P}_p$ is null
        $\hat{P}_p \leftarrow$ INSERT(D_k, δ_y) ▷ Insert at the beginning of the list
    else
        $\hat{P}_c \leftarrow$ INSERT$(D_k - \delta_x, \hat{P}_p.y_r)$
        $\hat{P}_c$.next $\leftarrow \hat{P}_p$.next
    $\hat{P}_p$.next $\leftarrow \hat{P}_c$
▷ Update intervals greater than D_k
while $\hat{P}_c$ not null
    SHIFT$(\hat{P}_c, \delta_x, \delta_y)$
    if $\hat{P}_p$ and $\hat{P}_c$ are in same region
        MERGE$(\hat{P}_p, \hat{P}_c)$
    $\hat{P}_p \leftarrow \hat{P}_c$; $\hat{P}_c \leftarrow \hat{P}_c$.next
end while

- When a new job j_k arrives in the system, we first check if the condition $\mathsf{dbi}(\Lambda, D_k) \geq E_k$ holds. Then, we insert it in B in absolute deadline order. For all the nodes in B with absolute deadline less or equal the new node, we simulate the UPONJOBDEADLINE-A operation on a duplicate list $\mathbb{L}'$. If any insert. merger of shift operation violates dbi, we reject j_k.

- When the deadline of job j_k in B elapses, UPONJOBDEADLINE-A is invoked, which performs UPDATE-A on list $\mathbb{L}$ and removes the node corresponding to j_k from the front of B.

- In the UPDATE-A operation, we first update all the intervals in $\mathbb{L}$ which have smaller length than the interval $D_k - \delta_x$ (Lines 2 to 5). Then we check if any approximation point $\hat{P}_i$ in the list has $D_k - \delta_x$ in between its anchors ($\hat{P}_i.x_l \leq D_k - \delta_x < \hat{P}_i.x_r$). A new point corresponding to j_k is inserted in between them if it is not within the redundancy region of $\hat{P}_i$. Finally in Lines 28 to 33, we update all the nodes in $\mathbb{L}$ which have interval lengths greater or equal $D_k - \delta_x$.

- Note that, while inserting a new node corresponding to j_k in $\mathbb{L}$, we ensure that it is inserted in the order of its interval length D_k. Further, we ensure that the demand corresponding to this interval is computed to represent the demand of all the jobs that have both arrival and deadline within this interval.

- Since the insertion point of j_k's interval D_k in $\mathbb{L}$ cannot be pre-determined (unlike the MAD case, where new interval is always inserted at the head of the list), the demand over D_k is not known when j_k arrives, and two different types of shift needs to be done based on whether the points in $\mathbb{L}$ are before or after D_k, it is not trivial for the admission controller to decide whether to accept j_k or not. For these reasons, we simulate the UPONJOBDEADLINE-A on a duplicate list $\mathbb{L}'$ for all active jobs that have deadline less or equal $\bar{d}_k$. If any operation violates the interface, we reject the job.

6.7.2 Algorithm Correctness

The correctness of APPROXIMATEAC-A is given by Theorem 16. The admission controller will not accept a job if any MERGE, SHIFT or INSERT on the duplicate list $\mathbb{L}'$ violates Equation 6.1 (Line 7). When it accepts a job, the interval corresponding to it will be inserted to $\mathbb{L}$ in absolute deadline (using list B). We show that after running APPROXIMATEAC-A and UPONJOBDEADLINE-A for each accepted jobs, there is a node in $\mathbb{L}$ for each interval, and demand over all intervals are correctly updated (similar to Lemma 46

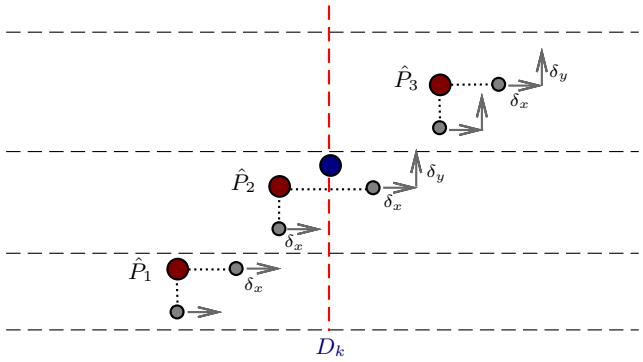


Figure 6.11: While inserting the interval D_k in $\mathbb{L}$, UPDATE-A shifts $\hat{P}_1$ by $(\delta_x, 0)$, $\hat{P}_3$ by (δ_x, δ_y). The anchors of $\hat{P}_2$ are updated according to their position.

and Lemma 47). As we use the same MERGE operation as in Algorithm 9, Lemma 45 also holds for this algorithm.

Theorem 16 *If* APPROXIMATEAC-A(Λ, j_k, ϵ) *returns "Accept", then* j_k *may be admitted without violating the demand-curve interface* Λ.

Lemma 56 *After the call to* APPROXIMATEAC-INIT() *and the* k*'th invocation of* APPROXIMATEAC-A-(Λ, j_k, ϵ) *where* $k \in \mathbb{N}$*, for aperiodic jobs* $j_1, j_2, \ldots, j_k$*, define* J_k *to be the set of jobs admitted by the approximate admission controller. For each job* $j_i \in J_k$*, after executing* UPONJOBDEADLINE-A(j_i)*, there exists an approximation point* $\hat{P}$ *in list* $\mathbb{L}$ *such that* $\hat{P}.x_l$ *is at most* $\bar{d}_{last} - A_i$ *and* $\hat{P}.y_r$ *is at least* $\mathsf{demand}(J_k, A_i, \bar{d}_{last})$*. Furthermore,* $\bar{d}_{last}$ *equals* $\max\left\{\{0\} \cup \{\bar{d}_i \mid j_i \in J_k\}\right\}$.

Proof: We prove the lemma by induction on k.

Base Case: When $k = 0$, APPROXIMATEAC-INIT() has been invoked and thus no jobs have been admitted; i.e, $J_0 = \emptyset$. The lemma is clearly true as $\mathbb{L}$ is initialized to $\emptyset$ and $\bar{d}_{last}$ is initialized to zero.

Inductive Hypothesis: Assume that the lemma holds for each i ($i = 1, 2, \ldots, k-1$) successive calls to APPROXIMATEAC-A(Λ, j_i, ϵ) and UPONJOBDEADLINE-A(j_i).

Inductive Step: When APPROXIMATEAC(Λ, j_k, ϵ) returns "reject", J_{k-1} is identical to J_k and $\bar{d}_{last}$ is not changed by any instruction in the execution path to "reject". Thus, by the inductive hypothesis, the lemma continues to hold as the state is identical to after the call to APPROXIMATEAC-A(Λ, j_k, ϵ).

Now, consider the case when APPROXIMATEAC-A(Λ, j_k, ϵ) returns "accept". After executing UPONJOBDEADLINE-A(j_k), line 3 of the procedure sets $\bar{d}_{last}$ equal to $\bar{d}_k$. Let the updated value of $\bar{d}_{last}$ and $\mathbb{L}$ be denoted by $\bar{d}^{\text{new}}_{last}$ and $\mathbb{L}^{\text{new}}$ respectively. Let $\bar{d}^{\text{old}}_{last}$ and $\mathbb{L}^{\text{old}}$ denote the value of the $\bar{d}_{last}$ and $\mathbb{L}$, prior to the invoke of UPONJOBDEADLINE-A(j_k). By the inductive hypothesis, for each job $j_i \in J_{k-1}$, there exists $\hat{P} \in \mathbb{L}^{\text{old}}$ such that $\hat{P}.x_l$ is at most $\bar{d}^{\text{old}}_{last} - A_i$ and $\hat{P}.y_r$ is at least $\mathsf{demand}(J_{k-1}, A_i, \bar{d}^{\text{old}}_{last})$. We have $\delta_x = \bar{d}^{\text{new}}_{last} - \bar{d}^{\text{old}}_{last}$ and $\delta_y = E_k$.

After the call to UPONJOBDEADLINE-A(j_k), a point corresponding to j_k is inserted in $\mathbb{L}$ in deadline order. UPDATE-A$(\delta_x, \delta_y, j_k)$ ensures that all the approximation points with $\hat{P}.x_r \leq D_k$ are to the left of the new point. The new point corresponds to the interval $[A_k, \bar{d}_k]$, and the demand over this interval is the demand for this job and demand for all the job $j_i \in J_k$ such that $A_i \geq A_k$ and $\bar{d}_i < \bar{d}_k$. This implies $\mathsf{demand}(J_k, A_k, \bar{d}_k) = E_k + \mathsf{demand}(J_k, \min_{i|A_i > A_k} A_i, \bar{d}_{last})$. If j_k is inserted in MAD order after j_{last}, i.e., $A_k \geq A_{last}$, $\mathsf{demand}(J_k, A_k, \bar{d}_k) = E_k$. Thus, for j_k after inserting point $\hat{P}$, $\hat{P}.x_l$ is at most $\bar{d}^{\text{new}}_{last} - A_k$ and $\hat{P}.y_r$ equals $\mathsf{demand}(J_k, A_k, \bar{d}^{\text{new}}_{last})$ (before any MERGE operation).

There are three cases we need to consider while performing UPDATE-A and subsequent MERGE operataions after inserting j_k in absolute deadline order in $\mathbb{L}$ as shown in Figure 6.11.

- **Case 1**. $\hat{P}.x_r < D_k - \delta_x$. This case represents all the points in $\mathbb{L}$ that correspond to an interval $[A_i, \bar{d}_i]$ such that $A_i \geq A_k$ and $\bar{d}_i < \bar{d}_k$ for $j_i \in J_k$ (node $\hat{P}_1$) in Figure 6.11). By definition of demand, in $[A_i, \bar{d}_k]$ interval, the demand will remain unchanged ($\delta_y = 0$). Thus we perform SHIFT$(\delta_x, 0)$ to update these points (Line 2 to 5). Note that, for these intervals, since the points are shifting only in X-axis to the right, no MERGE operation will be needed. The above proposition holds for these points.

- **Case 2**. $\hat{P}.x_l \geq D_k - \delta_x$. This case represents all the points in $\mathbb{L}$ that correspond to an interval $[A_i, \bar{d}_i]$ such that $A_i \leq A_k$ and $\bar{d}_i \leq \bar{d}_k$ (node $\hat{P}_3$) in Figure 6.11). We perform update similar to MAD in this case using SHIFT(δ_x, δ_y) for all approximation points to the right of $D_k - \delta_x$. Lemma 45, implies that if any approximation point $\hat{P}$ is deleted by a MERGE, then $\mathbb{L}^{\text{new}}$ has an approximation point $\hat{P}'$ with $\hat{P}'.x_l \leq \hat{P}.x_l$ and $\hat{P}.y_r \leq \hat{P}'.y_r$. Thus, each $\hat{P} \in \mathbb{L}^{\text{old}}$ is now $(\hat{P}.x_l + \delta_x, \hat{P}.y_r + \delta_y) \in \mathbb{L}^{\text{new}}$ or has an approximation point $\hat{P}'$ that covers it. Furthermore, $\hat{P}.x_l + \delta_x$ is at most $(\bar{d}^{\text{new}}_{last} - \bar{d}^{\text{old}}_{last}) + \bar{d}^{\text{old}}_{last} - A_i = \bar{d}^{\text{new}}_{last} - A_i$ and $\hat{P}.y_r + \delta_y$ is at most $\mathsf{demand}(J_{k-1}, A_i, \bar{d}^{\text{old}}_{last}) + E_i$. The last expression is equivalent to $\mathsf{demand}(J_k, A_i, \bar{d}^{\text{new}}_{last})$ since increasing the interval length by

δ_x includes only the new job j_k in the interval $[A_i, \bar{d}^{\text{new}}_{last}]$.

- **Case 3**. $\hat{P}.x_l \leq D_k - \delta_x < \hat{P}.x_r$. The last case ($\hat{P}_3$ in Figure 6.11) is when D_k falls in between the left and right anchor point of an approximation point $\hat{P}$ (node $\hat{P}_2$) in Figure 6.11). We shift left anchor similar to Case 1 (Line 7) and right anchor similar to Case 2 (Line 8). This ensures that $\hat{P}.x_l$ is atmost $\bar{d}^{\text{new}}_{last} - A_i$ and $\hat{P}.y_r$ includes the demand of the added job (δ_y). If the anchors are in the same region after shift, the new point corresponding to j_k will also be in the same region due to the fact that $\mathsf{demand}(J_k, A_k, \bar{d}_k) = \hat{P}.y_l + E_k \leq \hat{P}.y_r$ (after the shift of right anchor). In this case, the point corresponding to j_k will be within the redundancy region by Definition 19. Thus, no INSERT is needed in this case.

  Now, after the shift of the anchors, if they move to different regions, then we need to ensure that the above proposition holds. We split $\hat{P}$ into two points $\hat{P}_l$ and $\hat{P}_r$, and perform INSERT$(D_k, \hat{P}.y_l + E_k)$ into $\mathbb{L}$ between these points. For each of these points, the above proposition holds. Finally we check whether any two points among these three falls in the same region, in which case we perform MERGE on them and Lemma 45 ensures that each point in $\mathbb{L}$ is covered after the merge.

Therefore, for each of the cases the lemma holds for J_k. ■

The next lemma shows that the approximate admission controller returns "Accept" for a job j_k only when it is safe to do so.

Lemma 57 *After the call to* APPROXIMATEAC-INIT() *and the k'th invocation of* APPROXIMATEAC-A(Λ, j_k, ϵ) *where $k \in \mathbb{R}$, for aperiodic jobs $j_1, j_2, \ldots, j_k$, define J_k to be the set of jobs admitted by the approximate admission controller. It must be that Equation 6.1 holds for J_k and Λ.*

We can prove this lemma using similar techniques of Lemma 47. Theorem 16 immediately follows from the above lemma.

Essentially, the above approach uses B as a "staging" data structure to force nodes to be added to $\mathbb{L}$ in order of absolute deadline. To check whether we may admit a new job, we are essentially simulating the insert operation of the arrived active jobs in deadline order and checking to see if any interface violations occur. Thus, the above extension will add an additional multiplicative factor of N (i.e., the largest length of list B) to the overall time complexity. Future research is needed to determine whether this time complexity can be reduced further with a more sophisticated technique or data structure.

CHAPTER 7: ENFORCEMENT: APPLYING IN SLACK RECLAMATION

In this section we focus on efficiently determining runtime system slack to allocate unused processing resources at runtime while still enforcing the given subsystem interface. For such systems that can utilize slack, we are interested in finding an answer to the question: *given a compositional system where a component represents an* EDF*-schedulable sporadic tasks system [70], how much slack can be allowed at runtime to an active job such that the component is guaranteed to remain schedulable for any future job arrival scenarios*? To obtain optimal runtime system slack, along with the currently active jobs, we consider worst-case future job arrival scenario, and by *allowable* we mean future jobs will continue to meet their deadlines even if all subsequently scheduled jobs execute for their WCET. Although slack reclamation is well investigated in the literature for different system models, previous approaches either focused on more restrictive task models (e.g., strictly periodic tasks) or obtaining sufficient (but not necessary) lower bounds on slack. In contrast, our approach permits a more flexible task model and obtains optimal lower bound on slack at runtime i.e., if more slack is reclaimed beyond this lower bound, a deadline miss is obtainable if all future jobs execute to their WCET and subsequent future jobs are released with a minimum inter-arrival time separation.

> Given an interface Λ and a sporadic task system τ where τ is EDF-schedulable upon Λ in an "off-line" setting, our objective is to determine optimal system slack at any time T, such that the system remains schedulable for the current jobs and all subsequent jobs released by τ in any $T' \geq T$, that is, the interface is never violated.

With this goal in mind, our contributions in this chapter can be listed as follows:

- We characterize the optimal runtime system slack at any time T with the guarantee that all future jobs will remain schedulable and give a straightforward slack determination mechanism (Section 7.2).
- To address the spatial complexity of the above approach, we investigate the use of space-filling curves along with advance tree data structure (Section 7.3).

- We propose a simple approximation on the stored slack data to further reduce the space complexity in Section 7.4 and perform simulation to compare the proposed approaches (exact and approximate). We observe significant improvement in space complexity (Section 7.5).

7.1 Optimal Runtime Slack

We assume a compositional system with each component as a sporadic task system for this chapter (refer to Section 3.1.2 for a description of sporadic task system), and characterize optimal runtime slack for the component.

7.1.1 Task and Workload Models

Recall from Section 3.1.2 that a sporadic task τ_i is characterized by three parameters: worst-case execution time e_i, relative deadline d_i and minimum inter-arrival separation p_i. We assume constrained deadline sporadic tasks, i.e., $d_i \leq p_i$. Each task τ_i in the sporadic task system τ generates a potentially infinite sequence of jobs. Task system τ consists of n sporadic tasks.

We consider a discrete time system, and denote each discrete time point as T. At any time $T \in \mathbb{N}$, let $\tau_i(T)$ represent the "most-recently released job" of task τ_i with arrival time $a_i(T)$ and absolute deadline $\bar{d}_i(T)$. We say $\tau_i(T)$ is "active" if its deadline has not elapsed yet, that is, $a_i(T) \leq T < \bar{d}_i(T)$. Since we assumed $d_i \leq p_i$ for each task τ_i, the system consists at most one active job at any time T. Let $\tau_i(T)$ has remaining execution of $e_i(T)$ amount at time T where $e_i(T) \leq e_i$. If τ_i does not have any active job in the system ($T > \bar{d}_i(T)$), $e_i(T) = 0$.

At any time T, let $\tau'_i(T)$ represent the "earliest future job" of task τ_i. Since p_i is the minimum inter-arrival separation time among successive jobs, $\tau'_i(T)$ will have worst-case arrival time $a_i(T) + p_i$ and absolute deadline $\bar{d}_i(T) + p_i$. Let $o_i(T)$ represent the *release offset* of $\tau'_i(T)$ at time T, defined as follows.

$$o_i(T) \stackrel{\text{def}}{=} \begin{cases} a_i(T) + p_i - T, & \text{if } T - a_i(T) < p_i; \\ 0, & \text{Otherwise.} \end{cases} \tag{7.1}$$

This value represents earliest possible release time of $\tau'_i(T)$, with respect to the arrival time of most recently released job $\tau_i(T)$. When there is no active job of τ_i present in the system and $T \geq a_i(T) + p_i$, the offset $o_i(T)$ equals zero and $\tau'_i(T)$ can be released immediately. Let $o_\tau(T) = \{o_i(T)\}_{i=1:n}$ represent

the set of offset values for all $\tau_i \in \tau$ at time T.

§**Offline Workload Functions.** For determining schedulability of a sporadic task system at design time, it is often useful to quantify the maximum amount of execution that must complete over any given interval. We rephrase the definition of demand bound function from Section 3.2. We denote a time interval-length by t in contrast to a discrete time point denoted by T.

Definition 21 (Demand-Bound Function [21]) *For any time interval-length $t > 0$ and task τ_i, the* **demand-bound function** *(*dbf*) quantifies the maximum cumulative execution requirement of all jobs of τ_i that could have both an arrival time and deadline in any interval of length t. Baruah et al. [21] have shown that, for sporadic tasks,* dbf *can be calculated as follows.*

$$\mathsf{dbf}(\tau_i, t) \stackrel{def}{=} \max\left(0, \left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \cdot e_i. \tag{7.2}$$

Observe from the above definition that the dbf has discontinuities at time points of the form $t \equiv d_i + b \cdot p_i$ where $b \in \mathbb{N}_+$. Let $\mathsf{DBF}(\tau, t) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \mathsf{dbf}(\tau_i, t)$. The necessary and sufficient schedulability condition for a sporadic task system upon a preemptive uniprocessor platform of unit speed is given by the condition $\mathsf{DBF}(\tau, t) \leq t, \forall t : 0 \leq t \leq P(\tau)$ [21], where $P(\tau)$ is an upper bound on the maximum time instant that the schedulability condition must be verified at. $P(\tau)$ is at most the hyperperiod $H_\tau = \mathsf{lcm}_{\tau_i \in \tau}\{p_i\}$ (exponential) for the above model. A pseudo-polynomial bound is given by the following equation when $U_\tau < 1$ and $d_{max} = \max_{i=1:n}\{d_i\}$:

$$P(\tau) \stackrel{\text{def}}{=} \min\left[H_\tau, \max\left(d_{max}, \frac{1}{1-U}\sum_{i=1}^{n} U_i \cdot (p_i - d_i)\right)\right]. \tag{7.3}$$

For a compositional system with component interface specified by $\mathsf{dbi}(\Lambda, t)$, we rewrite the schedulability condition in Theorem 1 of Section 4.1 ([39]) when components are scheduled by EDF.

Theorem 17 (from [39]) *A sporadic task system τ is* EDF*-schedulable upon an arbitrary demand-interface Λ, if and only if,*

$$\mathsf{DBF}(\tau, t) \leq \mathsf{dbi}(\Lambda, t),\ \forall t \leq P(\tau) \tag{7.4}$$

where $P(\tau)$ equals $\mathsf{lcm}_{\tau_i \in \tau}\{p_i\} + \max_{\tau_i \in \tau}\{d_i\}$ when $U_\tau \leq 1$ and $P(\tau)$ is given by Equation 7.3 when $U_\tau < 1$.

This condition needs to be verified at each point in the testing set. As the size of the testing set is expo-

nential when $U_\tau \leq 1$ and pseudo-polynomial when $U_\tau < 1$ the complexity of the test is exponential and pseudo-polynomial respectively.

In general, the term "slack" for any time interval-length t represents the amount of processing resource that is not demanded (over any interval of length $\geq t$) by the tasks present in the system. A characterization of minimum slack for sporadic task system is given in [16].

Definition 22 (Minimum Offline Slack) *In a compositional system where a component C is characterized by an interface* $\mathsf{dbi}(\Lambda, t)$*,* minimum offline slack *represents the minimum of the difference between processing resource and workload demand over any interval of time.*

$$\mathcal{S}_\tau(t) \stackrel{def}{=} \min_{\delta \geq t} \{\mathsf{dbi}(\Lambda, \delta) - \mathsf{DBF}(\tau, \delta)\} \tag{7.5}$$

Equation 7.5 gives a lower bound on system slack. However, this function does not accurately capture system slack at runtime, since it does not account for already executed and remaining execution times for active jobs in the system.

§**Runtime Workload Functions.** To obtain an accurate characterization of *runtime system slack*, we now quantify the maximum *active* workload at any time T for any task τ_i.

Definition 23 (Active Job Demand) *The* **active job demand** *(*dbf-a*) of a task $\tau_i \in \tau$ quantifies the maximum execution requirement of the most recently released job $\tau_i(T)$ that is currently active in the system, i.e., $a_i(T) \leq T < \bar{d}_i(T)$.*

$$\mathsf{dbf\text{-}a}(\tau_i, T, t) \stackrel{def}{=} \begin{cases} 0, & \text{if } T > \bar{d}_i(T) \text{ and } \bar{d}_i(T) \leq T + t; \\ e_i(T), & \text{Otherwise.} \end{cases} \tag{7.6}$$

Let $\mathsf{DBF\text{-}A}(\tau, T, t) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \mathsf{dbf\text{-}a}(\tau_i, T, t)$ represent cumulative active demands. Recall that each $\tau_i \in \tau$ have at most one active job in the system since we assumed $d_i \leq p_i$.

Definition 24 (Projected dbf**)** *At any T, the* **projected demand-bound function** *(*dbf-p*) quantifies the maximum cumulative execution requirement of all jobs of τ_i that may have both an arrival time and deadline in the interval $[T, T + t]$, with respect to offset $o_i(T)$ of the earliest future job $\tau_i'(T)$.*

$$\mathsf{dbf\text{-}p}(\tau_i, o_i(T), t) \stackrel{def}{=} \max\left(0, \left\lfloor \frac{t - d_i - o_i(T)}{p_i} \right\rfloor + 1\right) \cdot e_i \tag{7.7}$$

Let $\mathsf{DBF\text{-}P}(\tau, o_\tau(T), t) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \mathsf{dbf\text{-}p}(\tau_i, o_i(T), t)$ be the cumulative projected demand-bound function. The intuition behind the above definition is, at any time T we can obtain the offset $o_i(T)$ of the earliest future job of τ_i and assume that the subsequent jobs will arrive as early as possible to incur worst-case demand in the future. Thus, at any time instant, projected demand bound function $\mathsf{dbf\text{-}p}(\tau_i, o_i(T), t)$ represents maximum demand for future jobs of τ_i within time interval $[T, T + t)$.

The difference between the projected demand bound function $\mathsf{DBF\text{-}P}$ and the demand-bound function DBF is that the former one is a function of current time T and varies with respect to the offset values (of earliest future job), and the later one represents demand over any interval assuming the worst-case (immediate) arrival time for all future jobs. Since our goal is to determine runtime system slack, the reasoning behind separating active demand ($\mathsf{dbf\text{-}a}$) and projected demand ($\mathsf{dbf\text{-}p}$) is analogous to separating dynamic part of the demand with static (or pre-computable) part of the demand. This will be more apparent in the next few sections.

7.1.2 Runtime System Slack

We are now prepared to extend the traditional notion of offline system slack for slack calculated at runtime.

Definition 25 (Runtime System Slack) *At any time T, the* minimum runtime system slack *for task system τ represents the minimum difference between the available processor time and the total potential demand over any time interval $[T, T + t)$. Formally,*

$$\mathcal{S}_\tau(T, t) \stackrel{def}{=} \min_{\delta \geq t} \{\mathsf{dbi}(\Lambda, \delta) - \mathsf{DBF\text{-}P}(\tau, o_\tau(T), \delta) - \mathsf{DBF\text{-}A}(\tau, T, \delta)\} \tag{7.8}$$

In the above definition, the processing resource represents the demand-interface $\mathsf{dbi}(\Lambda, t)$ of the task system (component). We obtain runtime slack at time T for any interval t by subtracting the projected demand and the active demand from the interface for all possible time intervals $\delta \geq t$. We can separate the pre-computable part of the above slack definition (i.e., the part that does not depend upon the active jobs). We will store these values for each task and $o_\tau(T)$ combination in a table called an *offset table* (as described in the next section).

$$\mathcal{S}_\tau^{static}(T, t) = \min_{\delta \geq t} \{\mathsf{dbi}(\Lambda, \delta) - \mathsf{DBF\text{-}P}(\tau, o_\tau(T), \delta)\} \tag{7.9}$$

By substituting Equation 7.9 into $\mathcal{S}_\tau(T,t)$, we can reformulate the calculation as:

$$\mathcal{S}_\tau(T,t) = \min_{\substack{\delta=\bar{d}_i(T)-T\ :\\ (\tau_i\in\tau)\wedge(T\le\bar{d}_i(T))\wedge(T+t\le\bar{d}_i(T))}} \{\mathcal{S}_\tau^{static}(T,\delta) - \mathsf{DBF\text{-}A}(\tau,T,\delta)\} \tag{7.10}$$

Note that, from Definition 23, active demand for task τ_i is only valid for intervals $t \le \bar{d}_i(T) - T$. This is reflected in the subscript of the above equation of $\mathcal{S}_\tau$.

Using the above definition and a simple extension of standard EDF-schedulability analysis [21], we may prove a necessary and sufficient test where an active job may utilize the system slack at time T.

Theorem 18 *Consider an active job $\tau_i(T)$ at time T of any task τ_i; job $\tau_i(T)$ may safely utilize the $\xi \ge 0$ units of system slack (e.g., by running ξ units beyond its WCET e_i), if and only if,*

$$\xi \le \mathcal{S}_\tau(T, \bar{d}_i(T) - T) \tag{7.11}$$

Proof: **(⇒)** We prove this direction by contrapositive. That is, assume that $\tau_i(T)$ runs ξ units beyond its WCET e_i such that $\xi > \mathcal{S}_\tau(T, \bar{d}_i(T) - T)$. We show that this will violate the demand interface $\mathsf{dbi}(\Lambda, t)$ for any $t \ge \bar{d}_i(T) - T$. In the worst case, all the tasks release their future jobs as early as possible i.e., the demand over the interval $\bar{d}_i(T) - T$ is given by $\mathsf{DBF}(\tau, \bar{d}_i(T) - T)$. Considering the overrun of $\tau_i(T)$, the total demand is $\mathsf{DBF}(\tau, \bar{d}_i(T) - T) + \xi$. By the definition of slack in Equation 7.8, there will be at least one time point $T' \ge \bar{d}_i(T)$ such that the cumulative demand of all the current and future jobs will be greater than the interface. Therefore, the interface will be violated.

(⇐) Taking contrapositive for this direction, assume that demand over any time interval $t \ge \bar{d}_i(T) - T$ increased such that $\mathsf{dbi}(\Lambda, t)$ is violated. We must show that $\xi > \mathcal{S}_\tau(T, \bar{d}_i(T) - T)$ for any $\tau_i(T)$ at time T. The maximum cumulative demand over interval $[T, T + \bar{d}_i(T)]$ is given by $\mathsf{DBF}(\tau, \bar{d}_i(T) - T)$. Since the demand over this interval is greater than $\mathsf{dbi}(\Lambda, t)$, from Equation 7.8 $\mathcal{S}_\tau(T,t) < 0$. Since $\xi > 0$, it must be that $\xi > \mathcal{S}_\tau(T,t)$. ■

Therefore, the slack defined above can be used to allow active jobs to safely overrun at most the slack amount along with the guarantee that all future jobs will meet their deadlines. Please note that once the system permits the overrun by ξ units; the remaining execution time of $\tau_i(T)$ should be increased by ξ to keep track of the overrun.

7.2 Determining Exact Runtime Slack

For a sporadic task system τ, at any time instant T, the worst-case runtime system slack depends on two factors: 1) the demand from the active jobs currently in the system (Equation 7.6); and 2) the demand from the future jobs (Equation 7.7). Considering these two factors, the obtained slack will be such that if any active job of τ_i overruns by at most the value obtained in Equation 7.11, the system will still remain schedulable in the future (Theorem 18). The first part of the slack depends on the demand from the active jobs in the system, and the second part can be precomputed using Equation 7.9. In the next section we describe how a slack table is computed.

7.2.1 Computing Slack Table

What is the online computational complexity of determining runtime system slack? If the system consists of n tasks, then for a given offset $o_\tau(T)$, there are only n active jobs in the system at time T. Thus, we can precompute $\mathcal{S}_\tau^{static}(T, \bar{d}_i(T) - T)$ for all $\tau_i \in \tau$. (Note that $\bar{d}_i(T) - T$ is entirely dependent upon the values of $o_\tau(T)$ and can also be determined offline). The offline complexity of the precomputation is pseudo-polynomial time (based on Equation 7.3). However, at runtime, we can use Equation 7.10 to do the computation online in $O(n)$ time using the precomputed offset table.

Consider the case when τ consists of a single task τ_i. At any time T, the most recently released job $\tau_i(T)$ can be either active ($T \leq \bar{d}_i(T)$) or completed execution ($T > \bar{d}_i(T)$). In the first case, the offset $o_i(T)$ for the earliest future job $\tau_i'(T)$ is $a_i(T) + p_i - T$ (Equation 7.1). Since $T > a_i(T)$ and time is discrete, we can have at most p_i (0 to $p_i - 1$) distinct offset values for $\tau_i'(T)$. If $\tau_i(T)$'s deadline has elapsed, i.e., there is no active job in the system, the next job can arrive immediately. That is, offset $o_i(T) = 0$. If we pre-compute static slack (Equation 7.9) for each of these distinct offset values assuming active demand $\mathsf{dbf\text{-}a} = 0$, we can easily determine the runtime slack for τ_i (Equation 7.10) using the active demand $e_i(T)$ of $\tau_i(T)$ at time T.

For n tasks, each with p_i distinct offset values, there are $\prod_{i=1}^{n} p_i$ distinct offset combinations. We denote these n-tuples as offset-tuple (i.e., $o_\tau(T)$). The idea is: we determine static slack for each task for a given offset n-tuple, using Equation 7.9, and store them in a table. Storing pre-computed slack values for all possible task offset combinations for each task requires an exponential size n-dimensional hypercube, where each dimension i represents the range of offset value for task τ_i. This data structure gives constant

lookup for the static slack values; however, the space requirement of the table is exponential. (We will address this drawback in the next section).

7.2.2 Exact Algorithm

The basic steps to compute slack $\mathcal{S}_\tau$ for any $\bar{d}_i(T) - T$ at time instance T are described below.

- **Iterate through all larger active-job interval lengths $\delta_j \geq d_j(T) - T$ for all $\tau_j : \bar{d}_j(T) \geq \bar{d}_i(T)$:** We determine slack for each interval as follows:

  1. **Lookup static slack from offset table**: For each δ_j and offset-tuple $o_i(T)$, we lookup in the slack table for corresponding slack-tuple.
  2. **Compute slack at each interval**: We determine slack at each δ_j using Equation 7.10. The active demand dbf-a at δ_j can be obtained via EDF scheduler's ready queue.

- **Obtain minimum slack** We take the minimum among all the slack values obtained for each $\tau_i \in \tau$ in the previous step to obtain minimum slack for interval $[T, \bar{d}_i(T))$.

The lookup can be done in constant time; while iterating through the larger active-job intervals and calculating dbf-a can be done in linear time. Thus, the total time complexity for determining runtime system slack is $O(n)$.

7.2.3 Example

The following example describes how we compute and populate the offset table. The cell values are denoted by "slack-tuple", and the cell index is denoted by "offset-tuple". Let $\tau \equiv \{\tau_1 = (1, 3, 4), \tau_2 = (1, 5, 6)\}$. For this example, the table shown in Table 7.1(b) is of size 24 ($p_1 \times p_2$). Cell (i, j) corresponds to offset combination (o_1, o_2) and contains minimum slack value at each active interval ($\bar{d}_i(T) - T$) of all the tasks. Cell $(0, 0)$ contains tuple $(2, 2)$ which represents slack for tasks τ_1 and τ_2 at active demand intervals of length 0 and 0. Cell $(5, 1)$ has slack tuple $(3, 3)$, this means for task τ_1, minimum slack at active demand interval length $o_1(T) - (p_1 - d_1) = 5 - (4 - 3) = 4$ is 3. Similarly, for τ_2, active demand interval is 0, since $o_2(T) < p_1 - d_1 = 2$, and minimum slack for the system is 3.

Table 7.1: Computation of offset table

(a) Tasks

Task	e_i	d_i	p_i
τ_1	1	3	4
τ_2	1	5	6

(b) Offset-table

	$o_1 = 0$	$o_1 = 1$	$o_1 = 2$	$o_1 = 3$
$o_2 = 0$	{2,2}	{3,3}	{3,3}	{4,4}
$o_2 = 1$	{2,2}	{3,3}	{4,4}	{4,4}
$o_2 = 2$	{2,2}	{3,3}	{4,4}	{5,5}
$o_2 = 3$	{2,2}	{3,3}	{4,4}	{5,5}
$o_2 = 4$	{2,2}	{3,3}	{4,4}	{5,5}
$o_2 = 5$	{3,2}	{3,3}	{4,4}	{5,5}

7.3 Efficient Implementation of Slack Table

The space complexity of the slack table is exponential, which makes it infeasible to use in practice. In this section we aim to improve the straight forward hypercube data structure to an index tree (B-tree). Further, to take the advantage of the spacial locality of data, we use a special mapping to convert the n-D offset-tuple to 1-D index using space filling curves.

7.3.1 Space-Filling Curves

A space-filling curve [77] is a way of mapping multi-dimensional space into one-dimensional space. The curve passes through every cell element in n dimensional space exactly once. Two widely-used curves are Hilbert curves and Z-order curves. The difference between different curves is in their order of mapping the data to one dimensional space. In Figure 7.1, these curves are shown, where for each type of curve a mapping function is used to convert the 2-D table to one dimensional curve, preserving the spacial locality of the data.

For the n-dimensional offset table, we first map each offset-tuple to a point in a space filling curve. Since the curve will preserve spacial locality, adjacent tuples with same slack-tuple value will be clustered in the curve. The overall space complexity is then reduced by taking distinct slack-tuples which are adjacent in space, that is, can be represented by a range in one dimension. We construct a B-tree from the mapped 1-D value of the offset-tuple. Each intermediate node in the tree represents mapped offset, and each leaf node contains slack-tuples. We only store distinct slack-tuples in the leaves, that is, for a range of same adjacent slack-tuples, we add a leaf for the first index. In the simulation section we show the space reduction using this approach.

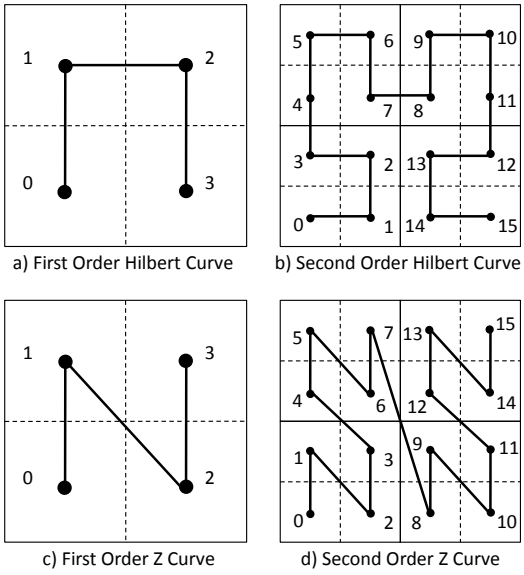


Figure 7.1: Space Filling Curves

To lookup an index in the B-tree, we first convert the lookup index (offset-tuple) to 1-D value. Then perform a lookup in the B-tree to find the data. The search complexity is $O(r + \lg N)$ in the worst case where r is size of the 1-D index in bits and N is the number of nodes in the tree. The size of N is bounded by $O(p_{max}^n)$, thus, the search complexity is $O(r + n \lg p_{\max})$, i.e., polynomial in the representation of the task system.

7.4 Determining Approximate Runtime Slack

To further reduce the space complexity of the offset-table data structure, we approximate the stored slack-tuples based on a given approximation parameter ϵ. Let $S(O) = \{S_i\}_{i=1:n}$ represent slack-tuple for the corresponding offset-tuple O (Section 7.1) in the offset table.

- Instead of grouping adjacent same-valued slack-tuples in the B-tree, we use approximation to obtain larger group of adjacent slack-tuples with same approximate value.

- For each dimension, we divide the range of slack values into distinct values $1, (1+\epsilon), (1+\epsilon)^2, (1+\epsilon)^3$ etc. For each slack-tuple, we map each of the slack values to a lower approximate value from previous line. In this way we obtain approximate slack-tuple where each values S_i within the range $(1+\epsilon)^k \leq S_i < (1+\epsilon)^{k+1}$ is mapped to the value $(1+\epsilon)^k$.

- Let $\hat{S}(O)$ represent the approximate slack for offset O. Then for each task, the slack value in the approximate slack tuple is computed as follows:

$$\hat{S}_i \stackrel{\text{def}}{=} \begin{cases} (1+\epsilon)^{\lfloor \log_{1+\epsilon} S_i \rfloor}, & \text{if } S_i \geq (1+\epsilon); \\ 0, & \text{Otherwise.} \end{cases} \tag{7.12}$$

- The approximation reduces the number of slack values from pseudo-polynomial to polynomial in n and $\frac{1}{\epsilon}$.

7.5 Simulation

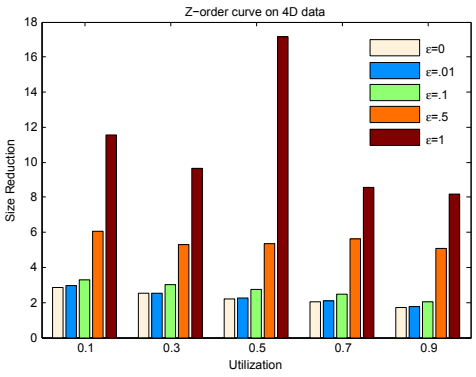
Since we have efficient time bounds on the runtime search operation, our evaluation focuses on the drawback of the proposed approach: space complexity and the error introduced in the approximation. We compare the size of the offset table data structure used in the exact and approximate tests. We synthetically generate sporadic tasks with random periods and utilizations, and compare the size of the offset table for different implementations. The parameters used for simulation are as follows:

- The system utilization U_τ is taken from the range $[0.1, 0.9]$ at 0.2-increments and individual task utilizations u_i are generated using UUniFast algorithm [25]. For a specific system utilization U_τ, this algorithm generates n random numbers in the range $[0, U_\tau]$ from uniform distribution which sum to U_τ.

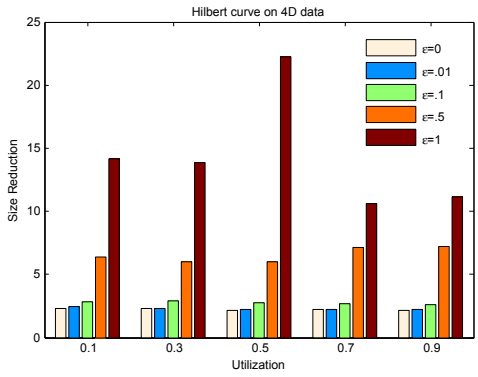
- Each sporadic task $\tau_i = (e_i, d_i, p_i)$ has a period p_i uniformly drawn from the interval $[5 - 20]$. (A small period range is used to keep H_τ from becoming too large). The execution time e_i is set to $u_i \times p_i$.

- We use $\epsilon = [0, 0.01, 0.1, 0.5, 1]$. Each of the point is an average of 10 simulation runs.

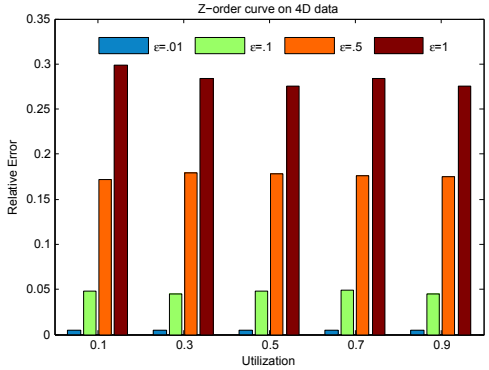
We measure three metrics for our simulation: 1) *size reduction*: ratio between the exact and the proposed data structure; 2) *average maximum relative error*: for each data point, we take the maximum of the ratio ($\frac{exact - approximate}{exact}$) among each dimension, and then take average of this value for all the data in the offset table; 3) *distinct tuple ratio*: ratio between the size of the offset table and the number of distinct tuple in the table. Size reduction shows the improvement of spatial complexity using space filling curves



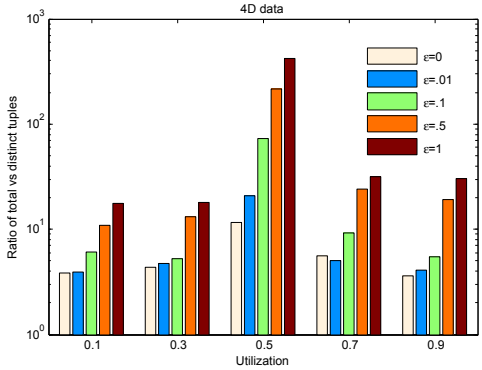
7.2.1: Size reduction using Z-order curve for 4D data



7.2.2: Size reduction using Hilbert curve for 4D data



7.2.3: Average maximum relative error in 4D data



7.2.4: Ratio between total tuples and distinct tuples in 4D data

Figure 7.2: Simulation results comparing space complexity of our approach

and B-tree data structure. The average relative error gives a comparison among the actual approximation in the data with the approximation parameter used to generate the data. Finally, the last metric depicts an upper bound on any size reduction.

In Figure 7.2.1 and 7.2.2, we compare the ratio between offset table size and the reduced data structure size for 4-D data ($|\tau| = 4$). For each system utilization in the range [0.1-0.9], we determine the size of the data structure for different values of ϵ. As expected, the size reduction is greater with higher values of ϵ. The data shows an order of magnitude improvement for $\epsilon = 1$. We observe that mapping using Hilbert curve gives better reduction in size than mapping using Z-order curve. As an example of absolute space reduction, the entire table for 4 tasks (without spacing filling curves) would be ≈6KB; for the Hilbert curve ≈1.8KB; and for $\epsilon = .5 \approx 0.9$KB.

The next plot (Figure 7.2.3) compares relative error in our approximations for 4D data. Observe that the average maximum relative is around one third to half, i.e., for $\epsilon = 0.1$ it is around 0.04, for $\epsilon = 1$, it is around 0.3. Finally, Figure 7.2.4 measures the ratio between the size of the offset table and number of distinct slack-tuples in it. For utilization $U_\tau = 0.5$, and $\epsilon = 0$, the ratio is around 10 where as the size reduction we obtained using both Z-order and Hilbert curve is around 2.5. This implies that the nature of the spatial locality of our data is not entirely captured by the existing curves that we have used. In future, we would like to improve upon this ratio.

7.6 Discussion

As argued in the Chapter 1, the runtime determination of slack is important for permitting tasks to extend their execution (e.g., in energy-aware systems). Towards this, we proposed an optimal runtime slack determination scheme which stores precomputed portions of slack in an offset table. We observed that the number of distinct tuples in the offset table is very small compared to the exponential size of the table and applied space-filling curves to reduce the space complexity. Although the space-filling curves we used in our experiments take advantage of spatially local data, they are unable to accurately capture the ranges of indices with similar (or approximately similar) data. In future, we will investigate the problem further to derive better space-filling curves to accurately capture spatial locality and obtain the optimal minimum space complexity. We also plan to compare our approach with existing solutions to slack reclamation problem in the context of EDF-scheduled sporadic task system.

The slack determination mechanism proposed in this chapter might be applied to predict allowable overrun in a system. For example, in mixed-criticality systems if the system slack is know at any time, this can be utilized to accommodate running lower criticality tasks at higher criticality mode. We could also potentially apply our approach to minimize the overhead and Cache Related Preemption Delay (CRPD) of popular preemptive scheduling for real-time systems. By efficiently computing the length of *non-preemption* region of execution for a job, the blocking overhead from critical tasks along with the preemption costs can be reduced.

CHAPTER 8: SUMMARY AND FUTURE WORK

In this chapter, we summarize our contribution in this thesis, and discuss potential future research directions from this work.

8.1 Our Contribution

In this thesis, we propose a thorough investigation of approximation algorithms for the minimization of interface bandwidth (MIB-RT) problem for server-based compositional frameworks. Previous work [83] has shown that exact algorithms for MIB-RT problem on periodic resources may require exponential time. Furthermore, current over-provisioned solutions have constant-factor approximation ratios, but cannot guarantee an approximation closer than a factor of $\frac{3}{2}$ larger than optimal. Therefore, to guarantee an arbitrary level of accuracy for the system designer, we propose a parametric approximation algorithm for the problem, which trades accuracy for computation time. We consider explicit-deadline periodic (EDP) resource model as real-time compositional framework for our approximation algorithm. For this model and any sporadic task system, our algorithm returns bandwidth that is at most a factor of $(1 + \epsilon)$ greater than the optimal minimum bandwidth, for any $\epsilon > 0$. Furthermore, it is shown that our algorithm is an FPTAS as it has time complexity that is polynomial in the number of tasks in the sporadic task system and the term $1/\epsilon$. Simulation results have shown that our approximation algorithm is effective at reducing the relative error over synthetically generated tasks, while maintaining a low runtime complexity.

For demand-based compositional frameworks, we address the problem of enforcing and policing the demand-curve interface for a subsystem of a compositional real-time system. We propose a simple demand interface model called the single-step demand interface (SSDI) and deriving efficient admission control algorithms for a subsystem with this demand-based interface. We provide a constant-time admission controller for jobs with MAD arrivals and generalize it for arbitrary aperiodic jobs. For a complex, arbitrary demand interface, we propose an exact admission control algorithm to police the subsystem load according to the interface, and show that for long-running online systems, this approach is infeasible. As an alternative we develop efficient polynomial time approximation algorithm for admission control. The development of these techniques should make it possible to utilize the rich theory developed for demand-

curve interfaces such as RTC and also ensure strong temporal isolation for this model. In our proposed algorithms, we have used list data structures to store intervals for the ease of presentation; the complexity of the approximate admission controller can be improved further by using advanced data structures (e.g. AVL tree).

Researchers applied compositional framework in avionics domain to schedule the ARINC 653 specied software components of the system. Our proposed bandwidth allocation algorithms can be applied in this setting which will signicantly speed up the analysis while exploring the design parameters in design phase. Further, another potential area of application may be in designing thermally constrained real-time systems, where the power-aware components dynamically tune their interfaces to meet the temporal and thermal constrains with the change in environmental temperature. For systems with large number of thermal operating modes, exact schedulability analysis will take long time, whereas, an approximate bandwidth allocation technique can help the system achieve this goal efciently.

Efficient policing and enforcement techniques for demand-curve interface models such as real-time calculus framework can be applied for online load reduction, QoS adaption etc. In the last part of this thesis, we focus on applying interface enforcement in efficient determination and reclamation of system slack. Given an interface, we give a novel technique using space filling curves to compute "allowable" system slack for a set of sporadic tasks at runtime. In designing an energy-aware system, the biggest challenge is to minimize energy consumption, thus, maximizing system idle time. Our proposed slack reclamation algorithm will help to decide whether the system will move to idle state or not by determining the energy savings from the slack period.

8.2 Future Work

In this section we briefly discuss future research directions from this doctoral dissertation.

8.2.1 Enforcing Demand-Curve Interface for Multiprocessor

As future work of arbitrary demand-interface model, we will extend these techniques to develop interface-policing policies for distributed and multiprocessor real-time systems. The multi supply function abstraction proposed by Bini et. al [26] and the improved parallel supply function abstraction [24] provide *virtual processor* abstractions for underlying physical multiprocessor platforms. These abstractions are equiva-

lent to a component interface of a compositional real-time system where components are scheduled upon a multiprocessor platform. For both of these abstractions, authors proposed sufficient time schedulability test which might be used for admission control of set of sporadic tasks as a component and scheduled by EDF. Given the abstractions for multiprocessor platforms, we can extend our admission controller for online systems with aperiodic workload and improve computational efficiency of the current approach.

8.2.2 Implementation of Admission Controller in Operating System

To evaluate the interface policing techniques, we plan to implement the proposed admission controllers (Chapter 6) in real-time operating systems (e.g., RTLinux). System will consist of several real-time applications, each specified by a demand-curve interface. Each application will have an associated admission controller which will enforce the demand-interface for dynamically arriving jobs in the system. In this way we will be able to evaluate the efficiency of the proposed admission controller for actual systems.

8.2.3 Further Extensions

Apart from the goals of our dissertation stated in the previous sections, we now discuss further research directions to the solutions obtained in our thesis. Since our online admission control algorithm does not have any assumption on the demand-interfaces, we can potentially use this approach to design real-time systems in an unpredictable environment where power or temperature may dynamically change along with the workload. In our admission control algorithms, we compute slack of demand from the interface and store minimum slack at any point of time. For power-aware systems, we can use this information to safely turn the CPU to a low power-mode for the duration of the slack period to save energy dynamically while still meeting the demand of the admitted jobs in the system.

We can apply our approach to minimize the overhead and Cache Related Preemption Delay (CRPD) of popular preemptive scheduling for real-time systems. By efficiently computing the length of *non-preemption* region of execution for a job at any given time in an online system, the blocking overhead from critical tasks along with the preemption costs can be reduced.

8.3 Concluding Remarks

Compositional design has a wide range of applicability from automotive to avionic systems [41]. Recent research is moving towards incorporating real-time frameworks to cyber-physical systems [40]. We envision that the compositional resource sharing frameworks will provide theoretical foundation for architectures like IMA, AUTOSAR etc. Further, component-based design with hierarchical scheduling allows establishing frameworks for real-time open environments in both uniprocessor and multiprocessor platforms. In this thesis we have addressed efficient resource allocation and interface enforcement problem in the context of compositional systems which is a very promising area of research.

We believe that the attainment of parametric approximation algorithms for MIB-RT problem under a variety of compositional frameworks will provide a real-time component designer with a valuable choice in determining how much interface bandwidth to trade for decreased speed-of-analysis. Extending the work contained in this thesis to more general task models (e.g., generalized multi-frame tasks [18], arbitrary deadline tasks) remains as future work. Furthermore, we expect that parametric approximation algorithms for MIB-RT on uniprocessor frameworks will also extend to multiprocessor compositional frameworks (e.g., [24, 26, 80]).

Our approach of enforcing arbitrary demand-interface would be invaluable for policing systems designed according to demand/supply-curve interfaces such as the real-time calculus models. As future work, we will extend these techniques to interface-policing techniques for distributed and multiprocessor real-time systems.

APPENDIX A

Proofs from Chapter 6

Proof of Lemma 39

Case ($T > \bar{d}_i$): By Lemma 37, $\phi(J, \Lambda, T)$ equals $\min\{\psi(J, \Lambda, A_k, T) \mid j_k \in J\}$. Thus, since $A_i \geq \max_{j_k \in J}\{A_k\}$ and $\bar{d}_i < T$, $\mathsf{demand}(J \cup \{j_i\}, A_k, T) = \mathsf{demand}(J, A_k, T) + E_i$ for all $j_k \in J$. Thus, by Equation 6.3, $\psi(J \cup \{j_i\}, \Lambda, A_k, T) = \psi(J, \Lambda, A_k, T) - E_i$. There is one new interval ending at T that must be considered according to Lemma 37: $[A_i, T]$. The value of $\psi(J \cup \{j_i\}, \Lambda, A_i, T)$ equals $\mathsf{dbi}(\Lambda, T - A_i) - E_i$ (there are no jobs other than j_i completely contained in the interval). These two observations on the value of ψ imply the second case of Equation 6.5.

Case ($T < \bar{d}_i$): Unlike the previous case, the demand over $[A_k, T]$ does not change under the addition of j_i as $T < \bar{d}_i$. Thus, $\psi(J \cup \{j_i\}, \Lambda, A_k, T)$ equals $\psi(J, \Lambda, A_k, T)$ for all $j_k \in J$. This implies the third case of Equation 6.5.

Case ($T = \bar{d}_i$): We must consider the value of ψ over all intervals $[A_k, \bar{d}_i]$ such that $j_k \in J \cup \{j_i\}$. For $[A_i, \bar{d}_i]$ clearly $\psi(J \cup \{j_i\}, \Lambda, A_i, \bar{d}_i)$ is $\mathsf{dbi}(\Lambda, D_i) - E_i$. For $[A_k, \bar{d}_i]$ such that $j_k \in J$, we first observe that $\bar{d}_i - A_k \geq \nu$. If this was not true, then $\mathsf{dbi}(\Lambda, D_k)$ equals zero, since $D_k \leq \bar{d}_i - A_k < \nu$. However, since $\phi(J, \Lambda, \bar{d}_k) \geq 0$, this implies that $0 \geq \mathsf{demand}(J, A_k, \bar{d}_k) \geq E_k$ which is a contradiction. Thus, by Equation 3.12, we may rewrite $\mathsf{dbi}(\Lambda, \bar{d}_i - A_k)$ as

$$
\begin{aligned}
&\sigma(\bar{d}_i - A_k - \nu) + \rho \\
&= \sigma(\bar{d}_{last}(J, j_i) - A_k - \nu) + \rho + \sigma(\bar{d}_i - \bar{d}_{last}(J, j_i)) \\
&= \mathsf{dbi}(\Lambda, \bar{d}_{last}(J, j_i) - A_k) + \sigma(\bar{d}_i - \bar{d}_{last}(J, j_i))
\end{aligned}
$$

since $\bar{d}_{last}(J, j_i) - A_k \geq D_k \geq \nu$. Using the new expression for $\mathsf{dbi}(\Lambda, \bar{d}_i - A_k)$ in ψ, we obtain

$$\begin{aligned}
&\psi(J \cup \{j_i\}, \Lambda, A_k, \bar{d}_i) \\
&= \mathsf{dbi}(\Lambda, \bar{d}_i - A_k) - \mathsf{demand}(J \cup \{j_i\}, A_k, \bar{d}_i) \\
&= \mathsf{dbi}(\Lambda, \bar{d}_{last}(J, j_i) - A_k) + \sigma(\bar{d}_i - \bar{d}_{last}(J, j_i)) \\
&\qquad -\mathsf{demand}(J \cup \{j_i\}, A_k, \bar{d}_{last}(J, j_i)) - E_i \\
&= \psi(J \cup \{j_i\}, \Lambda, A_k, \bar{d}_{last}(J, j_i)) + \sigma(\bar{d}_i - \bar{d}_{last}(J, j_i)) - E_i.
\end{aligned}$$

The second to last expression is due to the fact that j_i is the only job of $J \cup \{j_i\}$ that arrives after A_k but has deadline after $\bar{d}_{last}(J, j_i)$. Taking the minimum over all possible A_k values implies the first case of Equation 6.5. ■

Proof of Lemma 40 *Initialization:* Let us use the convention that j_0 is a dummy job that has deadline at time zero. Initially, $\mathsf{d} = 0$, $\mathsf{md} = \infty$, and the job set is empty. By Equation 6.2 of Definition 16, $\phi(\emptyset, \Lambda, T) = \infty$ for all $T > 0$ and the lemma initially holds.

Induction: Assume the lemma continues to hold after $i - 1$ invocations. By Lemma 39, we may calculate $\phi(\{j_1, j_2, \ldots, j_{i-1}, j_i\}, \Lambda, \bar{d}_i)$ by

$$\min \left\{ \begin{array}{l} \mathsf{dbi}(\Lambda, D_i) - E_i, \\ \phi(\{j_1, j_2, \ldots\}, \Lambda, \bar{d}_{i-1}) + \sigma(\bar{d}_i - \bar{d}_{i-1}) - E_i \end{array} \right\}.$$

As md equals $\phi(\{j_1, j_2, \ldots\}, \Lambda, \bar{d}_{i-1})$ and $\phi(\{j_1, j_2, \ldots\}, \Lambda, T) \geq 0$ for all $T > 0$ at the start of the i'th invocation and Line 1 update md according to the above expression and md satisfies the lemma. The variable d is set to $\bar{d}_i$. The only value that ϕ changes for is $T = \bar{d}_i$ (according to Lemma 39 and the fact there are no jobs with later deadline than j_i). Since Line 2 checks that $\mathsf{md} \geq 0$, this implies $\phi(\{j_1, j_2, \ldots, j_{i-1}, j_i\}, \Lambda, T) \geq 0$ for all $T > 0$. Thus, the lemma continues to hold after the invocation MAD-ADMISSIONCONTROL(j_i). ■

Proof of Lemma 41 The "only if" direction of the lemma is trivial as A_i and $\bar{d}_\ell$ are elements of $\mathbb{R}$ and $A_i \leq \bar{d}_\ell$. Thus, Equation 6.1 directly implies Equation 6.7.

For the "if" direction of the lemma, we will assume that Equation 6.7 is true, but Equation 6.1 is false

for some $T_1, T_2 \in \mathbb{R}$ where $0 \leq T_1 < T_2$; that is,

$$\mathsf{demand}(J, T_1, T_2) > \mathsf{dbi}(\Lambda, T_2 - T_1). \tag{1}$$

If $J = \emptyset$, then the demand over any interval is zero; since dbi is non-negative for all positive inputs, this leads to a contradiction of Equation 1. So, it must be that $J \neq \emptyset$. Let A_0 and d_0 denote zero and $A_{|J|+1}$ and $\bar{d}_{|J|+1}$ denote ∞. Consider two partitions of the interval $[0, \infty)$ into two sets of subintervals $(A_{i-1}, A_i]$ where $1 \leq i \leq |J| + 1$ and $[\bar{d}_\ell, \bar{d}_{\ell+1})$ where $0 \leq \ell \leq |J|$. Since $0 \leq T_1$, there exists some $i : (1 \leq i \leq |J| + 1)$ where $T_1 \in (A_{i-1}, A_i]$. Observe that $\mathsf{demand}(J, T_1, T_2) = \mathsf{demand}(J, A_i, T_2)$, since no jobs arrive in the interval (A_{i-1}, A_i). Similarly, there exists some $\ell : (0 \leq \ell \leq |J|)$ where $T_2 \in [\bar{d}_\ell, \bar{d}_{\ell+1})$ and $\mathsf{demand}(J, A_i, T_2) = \mathsf{demand}(J, A_i, \bar{d}_\ell)$. Thus, $\mathsf{demand}(J, T_1, T_2) = \mathsf{demand}(J, A_i, \bar{d}_\ell)$. By Equation 6.7, $\mathsf{demand}(J, T_1, T_2) \leq \mathsf{dbi}(\Lambda, \bar{d}_\ell - A_i)$ is true. Since $\bar{d}_\ell \leq T_2$, $T_1 \leq A_i$, and dbi is monotonically non-decreasing, it must be that $\mathsf{dbi}(\Lambda, \bar{d}_\ell - A_i) \leq \mathsf{dbi}(\Lambda, T_2 - T_1)$. These last two statements together imply that $\mathsf{demand}(J, T_1, T_2) \leq \mathsf{dbi}(\Lambda, T_2 - T_1)$ which contradicts Equation 1. Thus, the lemma is true. ■

Proof of Lemma 42 We prove the lemma by induction on k.

Base Case: When $k = 0$, EXACTAC-INIT() has been invoked and thus no jobs have been admitted; i.e, $J_0 = \emptyset$. The lemma is clearly true as S is initialized to $\emptyset$ and $\bar{d}_{last}$ is initialized to zero.

Inductive Hypothesis: Assume that the lemma holds for each i ($i = 1, 2, \ldots, k-1$) successive calls to EXACTAC$(\Lambda, \cdot)$.

Inductive Step: We must show that the lemma holds for the k'th call to EXACTAC(Λ, j_k). The admission controller can either return "accept" or "reject". Let us first consider the case that EXACTAC(Λ, j_k) returns "reject". Then, J_{k-1} is identical to J_k and $\bar{d}_{last}$ is not changed by any instruction in the execution path to "reject". Thus, by the inductive hypothesis, the lemma obviously continues to hold as the state is identical to after the call to EXACTAC(Λ, j_{k-1}).

Now, consider the case when EXACTAC(Λ, j_k) returns "accept". Line 13 of the procedure sets $\bar{d}_{last}$ equal to $\bar{d}_k$; Let the updated value of $\bar{d}_{last}$ and S be denoted by $\bar{d}^{\text{new}}_{last}$ and S^{new} respectively. Let $\bar{d}^{\text{old}}_{last}$ and S^{old} denote the value of the $\bar{d}_{last}$ and S variables, prior to EXACTAC(Λ, j_k). By the inductive hypothesis, for each job $j_\ell \in J_{k-1}$, there exists $(x, y) \in S^{\text{old}}$ such that x equals $\bar{d}^{\text{old}}_{last} - A_\ell$ and y equals $\mathsf{demand}(J_{k-1}, A_\ell, \bar{d}^{\text{old}}_{last})$. The *for-loop* of Lines 9 to 12 shifts each point (x, y) to the right by

$\delta_x = \bar{d}^{\text{new}}_{last} - \bar{d}^{\text{old}}_{last}$ and up by $\delta_y = E_k$. Thus, each $(x, y) \in S^{\text{old}}$ that corresponds to $j_\ell \in J_{k-1}$ is now $(x + \delta_x, y + \delta_y) \in S^{\text{new}}$. Furthermore, $x + \delta_x$ equals $(\bar{d}^{\text{new}}_{last} - \bar{d}^{\text{old}}_{last}) + \bar{d}^{\text{old}}_{last} - A_\ell = \bar{d}^{\text{new}}_{last} - A_\ell$ and $y + \delta_y$ equals $\mathsf{demand}(J_{k-1}, A_\ell, \bar{d}^{\text{old}}_{last}) + E_\ell$. The last expression is equivalent to $\mathsf{demand}(J_k, A_\ell, \bar{d}^{\text{new}}_{last})$ since increasing the interval length by δ_x includes only the new job j_k in the interval $[A_\ell, \bar{d}^{\text{new}}_{last}]$. Finally, adding the point $\{(D_k - \delta_x, E_k - \delta_y)\}$ in Line 2 and shifting by δ_x and δ_y is equivalent to adding (D_k, E_k) which equals $(\bar{d}^{\text{new}}_{last} - A_k, \mathsf{demand}(J_k, A_k, \bar{d}^{\text{new}}_{last}))$. Thus, the lemma holds for J_k. ■

Proof of Lemma 43 We prove the lemma by induction on k.

Base Case: When $k = 0$, EXACTAC-INIT() has been invoked and thus no jobs have been admitted; i.e, $J_0 = \emptyset$. Clearly, $\mathsf{demand}(J_k, T_1, T_2)$ is zero for all valid choices of T_1 and T_2. Therefore, Equation 6.1 is trivially true.

Inductive Hypothesis: Assume that Equation 6.1 holds for each i ($i = 1, 2, \ldots, k-1$) successive calls to EXACTAC$(\Lambda, \cdot)$. That is, J_i satisfies Equation 6.1. Furthermore, if job j_i is rejected, Equation 6.1 is false for $J_{i-1} \cup \{j_i\}$.

Inductive Step: We must show that, given the inductive hypothesis, 1) Equation 6.1 holds for J_k, and 2) if job j_k is rejected, then Equation 6.1 is false for $J_{k-1} \cup \{j_k\}$. During the call EXACTAC(Λ, j_k), the admission control can either return "accept" or "reject". Let us first consider the case that EXACTAC(Λ, j_k) returns "reject". Then, J_{k-1} is identical to J_k; since Equation 6.1 holds for J_{k-1}, by the inductive hypothesis, it obviously continues to hold for J_k. To see that Equation 6.1 is false for $J_{k-1} \cup \{j_k\}$, let S^{old} be the value of S before EXACTAC(Λ, j_k) and S^{new} be the value after the procedure call. δ_x is set to $\bar{d}^{\text{old}}_{last} - \bar{d}_k$ and δ_y is set to E_k. Observe that Line 4 must have evaluated to "true" for some $(x, y) \in S^{\text{old}} \cup \{(D_k - \delta_x, E_k - \delta_y)\}$. If (x, y) equals $(D_k - \delta_x, E_k - \delta_y)$, then $\mathsf{dbi}(\Lambda, D_k) < E_k$ which implies $\mathsf{dbi}(\Lambda, \bar{d}_k - A_k) < \mathsf{demand}(J_{k-1} \cup \{j_k\}, A_k, \bar{d}_k)$; in this case, Equation 6.1 is violated. Otherwise, if $(x, y) \in S^{\text{old}}$, then $\mathsf{dbi}(\Lambda, x + \delta_x) < y + \delta_y$ implies that $\mathsf{dbi}(\Lambda, \bar{d}_k - A_\ell) < \mathsf{demand}(J_{k-1} \cup \{j_k\}, A_\ell, \bar{d}_k)$ for some $j_\ell \in J_{k-1}$ by Lemma 42.

Now consider the case when EXACTAC(Λ, j_k) returns "accept". Lemma 42 implies that each $(x, y) \in S$ corresponds to $\left(\bar{d}_k - A_\ell, \mathsf{demand}(J_k, A_\ell, \bar{d}_k)\right)$ for some job $j_\ell \in J_k$. The fact that the j_k was accepted implies Line 4 was satisfied for each (x, y) and thus $x \geq y$; that is,

$$\forall j_\ell \in J_k, \mathsf{dbi}(\Lambda, \bar{d}_k - A_\ell) \geq \mathsf{demand}(J_k, A_\ell, \bar{d}_k). \tag{2}$$

The inductive hypothesis implies that prior to the invoke of EXACTAC(Λ, j_k), Equation 6.1 held; therefore,

$$\forall j_i, j_j \in J_{k-1} : \bar{d}_i \leq \bar{d}_j :: \mathsf{demand}(J_{k-1}, A_i, \bar{d}_j) \leq \mathsf{dbi}(\Lambda, \bar{d}_j - A_i). \tag{3}$$

Since $\bar{d}_k$ is later than any other $\bar{d}_i$ in J_{k-1}, j_k does not increase the demand in the above term $\mathsf{demand}(J_{k-1}, A_i, \bar{d}_j)$. Thus, the condition still holds when the left-hand side of the inequality is replaced with $\mathsf{demand}(J_k, A_i, \bar{d}_j)$. This observation regarding Equation 3 along with Equation 2 satisfy the supposition of Lemma 41 for the job set J_k; therefore, Equation 6.1 is satisfied for J_k. ■

Proof of Lemma 45 A node is deleted only in Line 9 or 13 of APPROXIMATEAC(Λ, j_k, ϵ) or in the MERGE subroutine. Clearly, in both cases, this only deletes the node that was inserted in Line 4; thus, by the note above this lemma, we do not consider this to be an approximation point. Therefore, the only subroutine that can delete nodes from $\mathbb{L}$ is the MERGE subroutine. Recall that when we merge approximation points $\hat{P}_1$ and $\hat{P}_2$, a new point $\hat{P}' = \left(\min_{k\in\{1,2\}}\{\hat{P}_k.x_l\}, \max_{k\in\{1,2\}}\{\hat{P}_k.y_r\}\right)$ is created. It is obvious that $\hat{P}_1$ and $\hat{P}_2$ are in the redundancy region $\mathcal{R}(\hat{P}')$ (refer to Figure 6.4(b)). Thus, for the two points deleted by the merge operation, by definition of redundancy region, $\hat{P}'.x_l \leq \hat{P}_k.x_l$ and $\hat{P}'.y_r \geq \hat{P}_k.y_r$ for $k \in \{1, 2\}$. ■

Proof of Lemma 47 We prove the lemma by induction on k.

Base Case: When $k = 0$, APPROXIMATEAC-INIT() has been invoked and thus no jobs have been admitted; i.e, $J_0 = \emptyset$. Clearly, $\mathsf{demand}(J_k, T_1, T_2)$ is zero for all valid choices of T_1 and T_2. Therefore, Equation 6.1 is trivially true.

Inductive Hypothesis: Assume that Equation 6.1 holds for each i ($i = 1, 2, \ldots, k-1$) successive calls to APPROXIMATEAC(Λ, j_i, ϵ). That is, J_i satisfies Equation 6.1.

Inductive Step: We must show that, given the inductive hypothesis, Equation 6.1 holds for J_k. During the call APPROXIMATEAC(Λ, j_k, ϵ), the admission control can either return "accept" or "reject". Let us first consider the case that APPROXIMATEAC(Λ, j_k, ϵ) returns "reject". Then, J_{k-1} is identical to J_k; since Equation 6.1 holds for J_{k-1}, by the inductive hypothesis, it obviously continues to hold for J_k.

Now consider the case that APPROXIMATEAC(Λ, j_k, ϵ) returns "accept". Lemma 46 implies that for each $j_\ell \in J_k$ there is an approximation point $\hat{P}$ with $\hat{P}.x_l \leq \bar{d}_k - A_\ell$ and $\hat{P}.y_r \geq \mathsf{demand}(J_k, A_\ell, \bar{d}_k)$.

The fact that the j_k was accepted implies Line 8 was satisfied for each approximation point; that is,

$$\forall j_\ell \in J_k, \mathsf{dbi}(\Lambda, \bar{d}_k - A_\ell) \geq \mathsf{demand}(J_k, A_\ell, \bar{d}_k). \tag{4}$$

The inductive hypothesis implies that prior to the invoke of APPROXIMATEAC(Λ, j_k, ϵ), Equation 6.1 held; therefore,

$$\forall j_i, j_j \in J_{k-1} : \bar{d}_i \leq \bar{d}_j :: \mathsf{demand}(J_{k-1}, A_i, \bar{d}_j) \leq \mathsf{dbi}(\Lambda, \bar{d}_j - A_i). \tag{5}$$

Since $\bar{d}_k$ is later than any other $\bar{d}_i$ in J_{k-1}, j_k does not increase the demand in the above term $\mathsf{demand}(J_{k-1}, A_i, \bar{d}_j)$. Thus, the condition still holds when the left-hand side of the inequality is replaced with $\mathsf{demand}(J_k, A_i, \bar{d}_j)$. This observation regarding Equation 5 along with Equation 4 satisfy the supposition of Lemma 41 for the job set J_k; therefore, Equation 6.1 is satisfied for J_k. ■

Proof of Lemma 48 Observe the only operations that change an approximation point are INSERT, SHIFT, and MERGE. We will show that, if the invariant initially holds for a point, the invariant will continue to hold after each operation. The INSERT operation creates a new approximation point with left anchor point equal to the right anchor point; thus, the invariant of the lemma initially holds. Now consider the SHIFT operation applied to an approximation point $\hat{P}$ where the invariant holds. The SHIFT operation is only called from Lines 2 and 5 of UPDATE. Let $\hat{P}'$ represent the approximation point after the application of SHIFT$(\hat{P}, \delta_x, \delta_y)$; thus, $\hat{P}'.y_r = \hat{P}.y_r + \delta_y$ and $\hat{P}'.y_l = \hat{P}.y_l + \delta_y$. If the invariant is true prior to the SHIFT call, then $\hat{P}'.y_r = \hat{P}.y_r + \delta_y \leq (1+\epsilon)\hat{P}.y_l + \delta_y \leq (1+\epsilon)(\hat{P}.y_l + \delta_y) = (1+\epsilon)\hat{P}'.y_l$. Thus, the invariant continues to hold after SHIFT.

Finally, to see that the invariant holds after MERGE, observe that the algorithm only merges two consecutive points $\hat{P}_\ell$ and $\hat{P}_{\ell+1}$ in the list $\mathbb{L}$ when both anchor points of these approximation points are completely within some region $\mathcal{A}^i$. For consecutive points $\hat{P}_\ell$ and $\hat{P}_{\ell+1}$ in $\mathbb{L}$, it may be shown (Lemma 51 in Section 6.3.3) that $\hat{P}_\ell.y_r \leq \hat{P}_{\ell+1}.y_r$. This observation taken together with the fact that the points are completely within $\mathcal{A}^i$ implies that $\mathcal{A}^i.lb \leq \hat{P}_\ell.y_l \leq \hat{P}_\ell.y_r \leq \hat{P}_{\ell+1}.y_r \leq \mathcal{A}^i.ub = (1+\epsilon)\mathcal{A}^i.lb$; the last inequality follows from Definition 17. Since $\mathcal{A}^i.lb \leq \hat{P}_\ell.y_l$, it must be that $\hat{P}_{\ell+1}.y_r \leq (1+\epsilon)\mathcal{A}^i.lb \leq (1+\epsilon)\hat{P}_\ell.y_l$. The MERGE operation will make a new approximation point $\hat{P}'$ with $\hat{P}'.y_l = \hat{P}_\ell.y_l$ and $\hat{P}'.y_r = \hat{P}_{\ell+1}.y_r$; thus, the invariant continues to hold for $\hat{P}'$. ■

Proof of Corollary 14 Lets denote the left anchor as P_L and right anchor as P_R. Given $P_L \in \mathcal{A}^i$ and $P_L \in \mathcal{A}^j$ where $j < i$, we prove the proposition by contradiction. Assume $j = i - 2$, that is, left anchor P_L is in region $\mathcal{A}^{i-2}$. Let q denote the distance of the anchor points in y-axes, that is $q = \hat{P}.yr - \hat{P}.yl$. By our assumption, q must be greater then the width of the region $\mathcal{A}^{i-1}$, i.e., $q > (1+\epsilon)^{i-2}$. Since approximation points with anchors in different regions are only created by shifting the points (upward and right), $\hat{P}$ must have been formed as an approximation point in any region $\mathcal{A}^{\ell}$, where $\ell \leq i-2$ (as $\hat{P}_L \in \mathcal{A}^{i-2}$). By definition of $(1+\epsilon)$-regions, the width of any such region is less or equal $(1+\epsilon)^{i-3}$. When an approximation point is created, both the anchors are within the same region, thus, the distance q in y-axes between $\hat{P}_L$ and $\hat{P}_R$ must be less or equal the width of region $\mathcal{A}^{i-2}$. This contradicts our assumption, since the relative distance between the anchors stays same after the shift operation which moved $\hat{P}_R$ to $\mathcal{A}^i$, and no merge is performed when anchors are in different regions . Therefore, we can conclude that $\hat{P}_L$ must reside in region $\mathcal{A}^{i-1}$. ■

Proof of Lemma 49 We prove the lemma by induction on k.

Base Case: When $k = 0$, APPROXIMATEAC-INIT() has been invoked and thus no jobs have been admitted; i.e, $J_0 = \emptyset$. Clearly, $\mathbb{L}$ is empty and the lemma is vacuously true.

Inductive Hypothesis: Assume after i ($i = 1, 2, \ldots, k-1$) successive calls to APPROXIMATEAC(Λ, j_i, ϵ) there is a corresponding exact interval for J_i for each left anchor point $\hat{P} \in \mathbb{L}$.

Inductive Step: If APPROXIMATEAC(Λ, j_k, ϵ) reject j_k, then the points of $\mathbb{L}$ are unaffected and J_{k-1} equals J_k; thus, by the inductive hypothesis, the lemma holds. If APPROXIMATEAC(Λ, j_k, ϵ) accepts j_k, then we must argue that each approximation point continues to have its left anchor point correspond to the exact demand of J_k over some interval. The only subroutines that could affect the approximation points of list $\mathbb{L}$ are INSERT, UPDATE, and MERGE. For INSERT, the newly-inserted point corresponds to the demand of job j_k. For UPDATE each point (including anchors) is shifted upwards $\delta_y = E_k$ and to the right by $\delta_x = \bar{d}_k - \bar{d}_{last}$. By the inductive hypothesis, for any $\hat{P} \in \mathbb{L}$, there exists $j_\ell \in J_{k-1}$ such that $\hat{P}.x_l = \bar{d}_{last} - A_\ell$ and $\hat{P}.y_l = \mathsf{demand}(J_{k-1}, A_\ell, \bar{d}_{last})$. Let $\hat{P}' \in \mathbb{L}$ be the resulting point after the SHIFT; thus, $\hat{P}'.x_l = \bar{d}_{last} - A_\ell + \bar{d}_k - \bar{d}_{last} = \bar{d}_k - A_\ell$ and $\hat{P}'.y_l = \mathsf{demand}(J_{k-1}, A_\ell, \bar{d}_{last}) + E_k$ which equals $\mathsf{demand}(J_k, A_\ell, \bar{d}_k)$; furthermore, $\bar{d}_{last}$ gets updated to $\bar{d}_k$. Thus, the lemma holds after a SHIFT. For MERGE, by inductive hypothesis and the fact the lemma holds after INSERT and SHIFT operations, the left anchors of the two points merged together correspond to the exact demand of J_k over some interval. By definition of merging, one of the left anchor points of the merged points becomes the new left anchor

of the new approximation point. Thus, the left anchor point continues to correspond to the exact demand over the same interval. ■

Proof of Lemma 50 A node corresponding to a newly-arrived job j_k arriving in MAD order will have the smallest interval length. (Recall that all existing intervals to be checked ending at $\bar{d}_k$. Due to MAD ordering property, $A_k \geq A_\ell$ for all $j_\ell \in J_k$, thus $[A_k, \bar{d}_k]$ is clearly the smallest interval). Since $\mathbb{L}$ is ordered in increasing x, a newly-inserted approximation point will always be inserted as a node at the front of list $\mathbb{L}$. ■

Proof of Lemma 51 Observe that the operations that change an approximation point are INSERT, SHIFT, and MERGE. We will show that, if the invariant initially holds for a point, the invariant will continue to hold after each of the operations. Obviously, the invariant holds when we have an initially empty list. Let us first consider the INSERT operation executed during the acceptance of some job j_k. By Lemma 50, we observed that a new node is inserted at the beginning of the list; furthermore, all subsequent approximation points (and their respective anchor points) are shifted upwards by $\delta_y = E_k$. Thus, since the newly created approximation point has a y-value of E_k for both anchor points and the point is entirely below the y-value of any other approximation point's anchor. The invariant will hold for the new point and every other point.

Now consider the SHIFT operation for any two approximation points $\hat{P}_1$ and $\hat{P}_2$. The SHIFT operation moves each anchor point of $\hat{P}_1$ and $\hat{P}_2$ upwards by the same amount; thus, the invariant continues to hold for $\hat{P}_1$ and $\hat{P}_2$ after the shift operation is applied to each approximation point in the list.

Finally, for the MERGE operation, consider two successive points $\hat{P}$ and $\hat{P}'$ in list $\mathbb{L}$ that are merged together to create $\hat{P}_1$. Consider a third point $\hat{P}_2$. If $\hat{P}_2$ appears later in the list, then prior to the call to MERGE (under the assumption that the invariant holds) we had $\hat{P}.y_l \leq \hat{P}.y_r \leq \hat{P}'.y_l \leq \hat{P}'.y_r \leq \hat{P}_2.y_l \leq \hat{P}_2.y_r$. After the call to MERGE, the approximation point $\hat{P}_1$ has left anchor point $(\hat{P}.x_l, \hat{P}.y_l)$ and right anchor point $(\hat{P}'.x_r, \hat{P}'.y_r)$. Thus, the invariant continues to hold for this case. The lemma may be shown symmetrically if P_2 precedes $\hat{P}$ and $\hat{P}'$. ■

Proof of Lemma 52 The nodes in $\mathbb{L}$ are ordered in non-decreasing y-value of right anchors by Lemma 51. The INSERT operation inserts new node at the beginning of the list by Lemma 50, the SHIFT operation shifts all the nodes same amount in X and Y-axes, and finally the MERGE operation merges consecutive nodes that are in same region. Clearly, MERGE will eliminate all but one approximation point for a $(1+\epsilon)$-region that may have temporarily contained more than one point. ■

REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 3–13, Madrid, Spain, December 1998. IEEE Computer Society Press.

[2] K. Albers, F. Bodmann, and F. Slomka. Advanced hierarchical event-stream model. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 211–220, Prague, Czech Republic, July 2008. IEEE Computer Society.

[3] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 187–195, Catania, Sicily, July 2004. IEEE Computer Society Press.

[4] L. d. Alfaro and T. A. Henzinger. Interface theories for component-based design. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 148–165, London, UK, 2001. Springer-Verlag.

[5] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: Response-time analysis and server design. In *Proceedings of the 4th ACM international Conference on Embedded Software*, pages 95–103, New York, NY, USA, 2004. ACM.

[6] J. Anderson, J. Calandrino, and U. Devi. Real-time scheduling on multicore platforms. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 179–190, April 2006.

[7] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, Cheju Island, South Korea, December 2000. IEEE Computer Society Press.

[8] B. Andersson. A preliminary idea for an 8-competitive, $\log_2 \mathrm{dmax} + \log_2 \log_2 1/u$ asymptotic-space, interface generation algorithm for two-level hierarchical scheduling of constrained-deadline sporadic tasks on a uniprocessor. *SIGBED Review*, 8:22–29, March 2011.

[9] B. Andersson and C. Ekelin. Exact admission-control for integrated aperiodic and periodic tasks. *Journal of Computer System and Sciences*, 73(2):225–241, October 2007.

[10] M. Asberg, M. Behnam, F. Nemati, and T. Nolte. Towards hierarchical scheduling in AUTOSAR. In *Proceedings of the 14th IEEE international Conference on Emerging Technologies & Factory Automation*, ETFA'09, pages 1181–1188. IEEE Press, 2009.

[11] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):285–292, 1993.

[12] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, England, 1991.

[13] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems*, 8:173–198, 1995.

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

[15] H. Aydi, P. Mejía-Alvarez, D. Mossé, and R. Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS '01, 2001.

[16] S. Baruah. Resource sharing in EDF-scheduled systems: A closer look. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 379–387, Rio de Janeiro, December 2006. IEEE Computer Society Press.

[17] S. Baruah. Component-based design of hard-real-time systems on multiprocessor platforms: Issues and ideas. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Barcelona, Spain, December 2008. IEEE Computer Society Press.

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

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

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

[21] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[22] M. Bertogna and S. Baruah. Limited preemption edf scheduling of sporadic task systems. *Industrial Informatics, IEEE Transactions on*, 6(4):579 –591, nov. 2010.

[23] E. Bini and S. Baruah. Efficient computation of response time bounds under fixed-priority scheduling. In *Proceedings of the 15th International Conference on Real-Time Systems*, pages 95–104, Nancy, France, March 2007.

[24] E. Bini, M. Bertogna, and S. K. Baruah. Virtual multiprocessor platforms: Specification and use. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 437–446, 2009.

[25] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 196–203. IEEE Computer Society, 2004.

[26] E. Bini, G. C. Buttazzo, and M. Bertogna. The multi supply function abstraction for multiprocessors. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 294–302, 2009.

[27] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 295 –304, 2000.

[28] S. Chakraborty, S. Knzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the 6th Design, Automation and Test in Europe (DATE)*, pages 190–195, March 2003.

[29] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10), October 1989.

[30] R. I. Davis. *On exploiting spare capacity in hard real-time systems*. PhD thesis, 1995.

[31] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 389–398, Miami, Florida, 2005. IEEE Computer Society.

[32] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9), September 2008.

[33] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM*, pages 109–120. Press, 2001.

[34] Z. Deng and J. Liu. Scheduling real-time applications in an Open environment. In *Proceedings of the 18th Real-Time Systems Symposium*, pages 308–319, San Francisco, CA, December 1997. IEEE Computer Society Press.

[35] F. Dewan and N. Fisher. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In *Proceedings of the 16th IEEE Real-Time Technology and Applications Symposium (RTAS)*, Stockholm, Sweden, 2010. IEEE.

[36] F. Dewan and N. Fisher. Admission control for real-time demand-curve interfaces. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Vienna, Austria, November 2011. IEEE Computer Society Press.

[37] F. Dewan and N. Fisher. Efficient admission control for enforcing arbitrary real-time demand-curve interfaces (extended version). Technical report, Wayne State University, 2012.

[38] A. Easwaran. *Compositional Schedulability Analysis Supporting Associativity, Optimality, Dependency and Concurrency*. PhD dissertation, University of Pennsylvania, Department of Computer and Information Science, 2007.

[39] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the IEEE Real-time Systems Symposium*, Tucson, Arizona, December 2007. IEEE Computer Society.

[40] A. Easwaran and I. Lee. Compositional schedulability analysis for cyber-physical systems. *SIGBED Rev.*, 5(1):1–2, 2008.

[41] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A compositional scheduling framework for digital avionics systems. *International Workshop on Real-Time Computing Systems and Applications*, 0:371–380, 2009.

[42] F. Eisenbrand and T. Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2010.

[43] X. A. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 26–35. IEEE Computer Society, 2002.

[44] N. Fisher. An FPTAS for interface selection in the periodic resource model. In *Proceedings of the 17th International Conference on Real-Time and Network Systems*, Paris, France, October 2009.

[45] N. Fisher and S. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 117–126, Palma de Mallorca, Balearic Islands, Spain, July 2005. IEEE Computer Society Press.

[46] N. Fisher and F. Dewan. Approximate bandwidth allocation for compositional real-time systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, Dublin, Ireland, July 2009. IEEE Computer Society Press.

[47] N. Fisher and F. Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real-Time Systems*, 48(3):223–263, 2012.

[48] T. A. Henzinger and S. Matic. An interface algebra for real-time components. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 253–266, Washington, DC, USA, April 2006. IEEE Computer Society.

[49] P. M. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi. The design and analysis of thermal-resilient hard-real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 67–76, Beijing, China, April 2012.

[50] R. Jejurikar and R. K. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Design Automation Conference*, pages 111–116. ACM.

[51] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, October 1986.

[52] W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proceedings of the conference on Design, automation and test in Europe*, pages 788–. IEEE Computer Society, 2002.

[53] P. Kumar, J.-J. Chen, and L. Thiele. Demand bound server: Generalized resource reservation for hard real-time systems. In *International Conference on Embedded Software (EMSOFT)*, Taipei, Taiwan, October 2011. ACM.

[54] T.-W. Kuo and C.-H. Li. A fixed prioriy driven open environment for real-time applications. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998. IEEE Computer Society Press.

[55] T.-W. Kuo and C.-H. Li. A fixed prioriy driven open environment for real-time applications. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998. IEEE Computer Society Press.

[56] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989*, pages 166–171, Santa Monica, California, USA, December 1989. IEEE Computer Society Press.

[57] J. Lehoczky, L. Sha, and J. Stronider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the Real-Time Systems Symposium*, pages 261–270, San Jose, CA, December 1987. IEEE.

[58] J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[59] H. Leontyev and J. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008. IEEE Computer Society Press.

[60] H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Real-Time Systems*, 43(1):60–92, 2009.

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

[62] G. Lipari and S. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 166–175, Washington, DC, May–June 2000. IEEE Computer Society Press.

[63] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the EuroMicro Conference on Real-time Systems*, pages 151–160, Porto, Portugal, 2003. IEEE Computer Society.

[64] G. Lipari and G. Buttazzo. Scheduling real-time multi-task applications in an open system. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, York, UK, June 1999. IEEE Computer Society Press.

[65] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments. In *Proceedings of the Real-Time Systems Symposium*, Orlando, FL, November 2000. IEEE Computer Society Press.

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

[67] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. Iris: a new reclaiming algorithm for server-based real-time systems. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 211 – 218, may 2004.

[68] A. Mok. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems*, pages 42–46, Washington D.C., May 1988.

[69] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, pages 75–84. IEEE, May 2001.

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

[71] C. Okwudire, M. van den Heuvel, R. Bril, and J. Lukkien. Exploiting harmonic periods to improve linearly approximated response-time upper bounds. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, September 2010.

[72] R. Pellizzoni and M. Caccamo. M-cash: A real-time resource reclaiming algorithm for multiprocessor platforms. *Real-Time System*, 40(1):117–147, Oct. 2008.

[73] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35(5):89–102, Oct. 2001.

[74] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Readings in multimedia computing and networking*, pages 476–490. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[75] J. Regehr and J. A. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 3–14, London, UK, December 2001.

[76] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 173–181, Vienna, Austria, June 2002. IEEE Computer Society Press.

[77] H. Sagan. *Space-Filling Curves*. Springer, 1 edition, Sept.

[78] T. seng Tia, J. W.-S. Liu, J. Sun, L. Jun, and R. Ha. A linear-time optimal acceptance test for scheduling of hard real-time tasks, 1994.

[79] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 181–190, 2008.

[80] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008. IEEE Computer Society Press.

[81] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 2–13. IEEE Computer Society, 2003.

[82] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 57–67. IEEE Computer Society, 2004.

[83] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3), April 2008.

[84] G. H. Software. ARINC 653 partition scheduler. Technical report. In www.ghs.com/products/safety critical/arinc653.html.

[85] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of the Real-Time Systems Symposium*, pages 251–258, Huntsville, Alabama, December 1988. IEEE.

[86] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1:27–69, 1989.

[87] J. Strosnider, J. Lehoczky, and L. Sha. The deferrable server algorithm for enhancing aperiodic responsiveness in hard-real-teime environments. *IEEE Transactions on Computers*, 44(1), January 1995.

[88] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 101–104, 2000.

[89] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software*, pages 34–43, New York, NY, USA, 2006. ACM.

[90] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 6:133–151, 1994.

[91] M. M. H. P. van den Heuvel, P. J. L. Cuijpers, J. J. Lukkien, and N. Fisher. Revised budget allocations for fxed-priority-scheduled periodic resources. Computer science reports, Technische Universiteit Eindhoven (TU/e), 2012.

[92] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin-Heidelberg-New York-Barcelona-Hong Kong-London-Milan-Paris-Singapur-Tokyo, 2001.

[93] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of the 5th ACM international Conference on Embedded Software*, pages 80–89, New York, NY, USA, 2005. ACM.

[94] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *IEEE Real-Time Technology and Applications Symposium*, pages 243–252, 2006.

[95] W. Wang, S. Ranka, and P. Mishra. Energy-aware dynamic slack allocation for real-time multitasking systems. *Sustainable Computing: Informatics and Systems*, 2(3):128 – 137, 2012.

[96] Windriver. Platform for ARINC 653. Technical report. In www.windriver.com/products/platforms/safety_critical.

ABSTRACT

EFFICIENT ALLOCATION AND ENFORCEMENT OF INTERFACES IN COMPOSITIONAL REAL-TIME SYSTEMS

by

FARHANA DEWAN

May 2014

Advisor: Dr. Nathan Fisher

Major: Computer Science

Degree: Doctor of Philosophy

Compositional real-time research has become one of the emerging trends in embedded and real-time systems due to the increasing scale and complexity of such systems. In this design paradigm, a large system is decomposed into smaller and simpler *components*, and developed independently. To hide internal complexity from the entire system, components abstract their temporal requirements via *interfaces*. Such systems are mostly implemented by *resource partitions* which ensure that the components receive resources according to their interfaces. Potential implementations of a resource partition are via *server-based interfaces* or *demand-based interfaces*. In the former model, access to the processing resource for each component is provided by a server, which ensures that the real-time requirements are satisfied ensuring temporal isolation among components (e.g., one component is isolated from the potentially faulty temporal behavior of another component). In the later model, the component demand is accurately modeled by a demand-curve interface and resource is provided to the component according to the interface.

In this dissertation our goal is to address the following thesis:

Currently, server-based interfaces ensure strong temporal isolation among components at the cost of resource over-provisioning whereas demand-based interfaces precisely model the resource demand of a component without the guarantee of temporal isolation. For both these models, we show that efficient and effective resource allocation as well as strict temporal isolation among components can be achieved. Specifically, we obtain efficient and near-optimal bandwidth allocation schemes and admission controllers for periodic resource model and

arbitrary demand-based interface respectively. Furthermore, efficient slack reclamation technique can be obtained to allocate unused processing resources at runtime while still enforcing the given interface.

State-of-the-art bandwidth allocation algorithms for server-based models use either exponential-time or pseudo-polynomial-time techniques for exact allocation, or linear-time, utilization-based techniques which may over-provision bandwidth. In this thesis, we propose research into a third possible approach: parametric approximation algorithms, with the goal to allow a component designer to pre-specify an arbitrary level of accuracy for bandwidth computation, and trade increased accuracy for increased computational complexity. For server-based interfaces, we address efficient resource allocation among components by proposing fully-polynomial-time approximation schemes (FPTAS) for allocating processing resource to components scheduled by earliest-deadline-first (EDF) or fixed-priority (FP) scheduling algorithm. Our algorithms take, as parameters, the task system and an accuracy parameter $\epsilon > 0$, and return a bandwidth which is guaranteed to be at most a factor $(1 + \epsilon)$ more than the optimal minimum bandwidth required to successfully schedule the task system. Furthermore, the algorithm has time complexity that is polynomial in the number of tasks and $1/\epsilon$. Via simulation, we show that our algorithms are quite accurate over synthetically generated task systems even for medium-sized values of ϵ.

Demand-based interfaces provide a precise characterization of the processing resource demanded by a component using fine-grained approaches such as Real-Time Calculus. Although such interfaces must be strictly enforceable to be capable of guaranteeing temporal isolation among safety critical applications, the model does not guarantee temporal isolation among components. We address the lack of interface-policing protocols, and propose both exact and near-optimal admission control algorithms for components specified by demand-interfaces. Given a simple demand-interface, our algorithm runs in constant time for *monotonic absolute deadline* (MAD) jobs and $O(\log N)$ time for arbitrary aperiodic jobs where N is the number of active jobs in the system. For arbitrary demand-curve interfaces, we propose exact admission control scheme for aperiodic workload and show that it is computationally infeasible for long-running online system. For this setting we propose a parametric approximate admission control algorithm, which has polynomial time complexity in terms of number of active jobs in the system and the approximation parameter ϵ. We implement each of our proposed admission controllers and show that our approximate approach is both efficient and precise in comparison to the exact approach via simulation.

In traditional hard real-time systems, worst-case analysis is used to determine the schedulability of the system. However, the runtime behavior of the system is often unpredictable. Worst-case execution time (WCET) overestimates resource demand since the actual execution time rarely approaches the worst-case estimate; jobs that do not execute to their WCET leave some "slack" in the system that can potentially be reclaimed. In the final part of the thesis, our goal is to address the challenge of determining at runtime the maximum allowable slack for any job in a system of sporadic real-time tasks while ensuring system schedulability. We provide a novel technique to optimally and efficiently determine system slack at runtime considering already arrived jobs and the worst-case job arrival sequences in the future. Our generic solution is based on slack computation from the system supply and can be potentially applied to various systems e.g., energy-aware systems, mixed-criticality systems, limited-preemption scheduling etc.

AUTOBIOGRAPHICAL STATEMENT

Farhana Dewan, the second child of Lutfor Rahman Dewan and Anwara Begum, was born in Bangladesh, a small green country in South Asia. She grew up in Dhaka and went to Lalmatia Girls High School for primary and secondary education. She passed higher secondary from Holy Cross College, Dhaka in 2000. She completed Bachelor's in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka in 2006. She worked in a research firm Commlink Info Tech Ltd in Dhaka for two years. She left Bangladesh for higher studies in the United States in August 2008.

Farhana entered the Graduate School of Wayne State University at Detroit, Michigan in Fall 2008 to pursue Ph.D. in Computer Science. She received her Master's degree in Computer Science in May 2012. While pursuing her Ph.D. under the supervision of Dr. Nathan Fisher, she was awarded Graduate Research Assistantship for five academic years 2008-2013 while working at the COmpositional PArallel Real-Time Systems (CoPaRTS) lab. Her research concentration is on compositional real-time systems, approximation algorithms, distributed systems and energy-aware systems. During this period she published her research in reputed real-time systems conferences and journals. She received best student paper award for her paper in the primer conference in real-time systems, the 33rd IEEE Real-Time Systems Symposium. She also received award for distinguished achievement in graduate student research from College of Engineering, Wayne State University.

Farhana started working for Microsoft, Corp. in January 2014. She briefly worked with the OnStar team of General Motors in 2013. She is a student member of ACM, IEEE and affiliated with Women in Computing (WinCS) group of Wayne State University.