Schedulability analysis and low-criticality execution support for mixed-criticality real-time systems on uniprocessors

Gu, Xiaozhe

2018

Gu, X. (2018). Schedulability analysis and low-criticality execution support for mixed-criticality real-time systems on uniprocessors. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73157>

<https://doi.org/10.32657/10356/73157>

**SCHEDULABILITY ANALYSIS AND LOW-CRITICALITY
EXECUTION SUPPORT FOR MIXED-CRITICALITY
REAL-TIME SYSTEMS ON UNIPROCESSORS**

XIAOZHE GU

School of Computer Science and Engineering

A dissertation submitted to the Nanyang Technological University in partial fulfilment of the
requirement for the degree of Doctor of Philosophy

2017

Acknowledgements

I have always been feeling lucky to get the chance to pursue the doctoral degree at Nanyang Technological University. During the pleasant four years time as a Ph.D student, I have received a lot of help and support from many people both in research and daily life without which this dissertation could not have been completed. I wish to express my sincere thanks to these people for their kindness.

To the uttermost importance, I would like to express my sincere gratitude to my advisor Arvind Easwaran for his continuous support of my Ph.D study and research during the past four years. It is him who guided me into the door of Computer Science research and showed me the charm of an academic life. I can still clearly remember the first days at NTU when he enlightened me about real-time scheduling theory. His patient guidance and knowledgeableness benefited me a lot during my Ph.D life.

I would like to extend my heartfelt appreciation to members of the thesis advisory committee and the qualification examination committee: Prof Anupam Chattopadhyay, Prof Sanjoy Baruah, Prof He Bingsheng, Prof Vun Chan Hua Nicholas and Prof Hartmut Klauck, for the time and effort they spent and the valuable suggestions that I received. It is my great fortune to acquire guidance and advice from them.

I want to thank all the colleagues and friends in CPS-research-group and Hardware and Embedded Systems Lab (HESL), Saravanan Ramanathan, Sriram Vasudevan, Sundar Vijaya Kumar, Luo Tao, Li Xiangwei, Chen Hao, Jiang Lianlian, Mao Fubing and so many others, for the past four years of good time. I really learnt a lot from you, and the delightful memory will never fade away.

I would also like to thank my parents who gave birth to me and love me unconditionally. My parents' trust and support is a significant motivation for me to seek the life I want. Without their inspiration, I could not believe I can complete my Ph.D study at NTU.

Abstract

An increasing trend in safety-critical real-time systems is towards open computing environments, where applications with different degrees of importance are developed independently and integrated together on a single computing platform. In these systems, the correct behavior of some application (e.g., flight control) is more important (“critical”) to the overall safety of the system than that of another (e.g., in-flight cooling). Such a system that consolidates multiple functionalities of different importance or criticality is called a mixed-criticality system.

Classical real-time scheduling theory can be used to guarantee the correctness of mixed-criticality systems, such that applications at different levels of criticality are verified with the corresponding degree of assurance. However, conservative resource reservations are often used to achieve this correctness at high assurance levels, and this results in severe under-utilization of system resources. To address this problem, real-time scheduling theory for mixed-criticality systems has been developed. The primary objective of this theory is to improve resource utilization, while still guaranteeing the safe execution of critical applications. Focusing on mixed-criticality scheduling theory, this dissertation addresses two important problems in the design of scheduling algorithms for mixed-criticality systems on uniprocessor platforms.

First, we focus on the design of scheduling algorithms and analysis techniques for mixed-criticality systems that can successfully schedule a large proportion of such systems. Since mixed-criticality scheduling has been shown to be a NP-Hard problem, it is important to develop techniques that improve schedulability while still having efficient runtimes. To address this problem, we propose a demand bound function (DBF) based schedulability analysis technique for earliest deadline first (EDF) scheduling of mixed-criticality systems, and also develop a corresponding scheduling strategy. This analysis considers the different execution modes in a mixed-criticality system collectively, and is thus able to overcome some of the pessimistic assumptions in existing analysis. The proposed scheduling strategy is thus able to achieve much better schedulability, particularly for systems with more than two criticality levels, without any increase in the runtimes.

Second, we also propose novel scheduling models and strategies to support the execution of less critical applications in mixed-criticality systems. Several of the studies in mixed-criticality make simplifying, but unrealistic, assumptions about the need to execute less critical applications. They assume that when critical applications demand additional system resources, it is safe to completely suspend the less critical applications in order to ensure a safe execution of the critical ones. Furthermore, many of these studies also assume that the critical applications are likely to demand these additional resources at the same time, and hence consider it reasonable to reduce support to all the less critical applications. These assumptions are not only impractical, but also lead to under-utilization of system resources.

To contain the impact on less critical applications, we first propose a component-based scheduling model and extend the above DBF-based schedulability analysis technique for the same. In this model, a component boundary isolates the critical applications within the component from the less critical ones in other components, as long as the demand for additional resources is within defined limits. Thus, the model provides service differentiation among the less critical applications depending on the demand of individual critical applications at any point in time. To further support the execution of less critical applications, we also propose a dynamic task and EDF based scheduling model under which the budgets for individual critical applications are determined at runtime. Thus, by dynamically distributing the total available budget among critical applications, we are able to significantly postpone the time instant when the demand from critical applications exceeds defined limits. As a consequence, the less critical applications in the system are able to get full service for a prolonged duration.

Table of Contents

Acknowledgments	I
Abstract	II
List of Figures	VIII
List of Tables	XI
List of Abbreviations	XII
List of Publications	1
1 Introduction	2
1.1 Real-Time System	2
1.1.1 Hardware platforms	4
1.2 Mixed-Criticality System (MCS)	4
1.2.1 Mixed-Criticality Scheduling	5
1.3 Motivation and Problem Statement	6
1.4 Contributions and Organization	8
2 Background	10
2.1 Real-Time Scheduling Theory	10
2.1.1 Sporadic Task Model	10
2.1.2 Analysis for Earliest-deadline-first (EDF) Scheduling	11
2.2 MC Task and Scheduling Model	12
2.2.1 Vestal's MC Model	13
2.2.2 Dual-Criticality Model	14
2.2.3 Execution Semantics of Vestal's Model	14

TABLE OF CONTENTS

2.2.3.1	Budget and WCET Estimation	16
2.3	Brief Introduction of EDF-VD*	17
3	Related Work	19
3.1	Scheduling Policies and Schedulability Tests	19
3.1.1	Fixed-Priority Scheduling	19
3.1.2	EDF Scheduling	21
3.2	Support for the Execution of LC Tasks	22
3.2.1	Techniques that provide degraded QoS for LC tasks in HC Mode	23
3.2.2	Techniques to limit the impact of mode-switch	24
3.2.3	Techniques to postpone mode-switches	25
3.2.4	Techniques to return to LC Mode	26
4	Efficient Schedulability Test for EDF-VD* Scheduling Policy	28
4.1	Motivation: Limitations of existing DBF-based analyses	29
4.2	SM DBF-Based Test	30
4.2.1	Formulating SM DBF	31
4.3	Multi-Mode DBF-Based Test For MC Task Systems	32
4.3.1	MM DBF for Jobs	34
4.3.2	MM DBF for a Single Task ($dbf_{MM}(\tau_i, t^*(m), t, m)$)	37
4.3.2.1	Case When $L_i = m - 1$	37
4.3.2.2	Case When $L_i \geq m$	41
4.3.3	MM DBF-based Schedulability Test	43
4.3.3.1	Upper Bound for t^{max}	45
4.4	Virtual Deadline Assignment Algorithm	46
4.4.1	Algorithm 1: TuneMode(m)	46
4.4.2	Algorithm 2: TuneSystem(τ)	47
4.4.3	Algorithm 3: find_candidate_m(Ψ, t)	49
4.5	Evaluation	50
4.5.1	Task Set Generation	51
4.5.2	Evaluation of Dual-Criticality Systems	51

TABLE OF CONTENTS

4.5.3	Evaluation of Multi-Criticality Systems	52
4.6	Conclusions and Future Work	54
5	Component-based Technique for Supporting Low-Criticality Executions	58
5.1	Motivation and Contributions	59
5.2	System Model	60
5.2.1	Task and Component	60
5.2.2	Execution Model	61
5.3	Schedulability Test for Flat Scheduling Framework	63
5.3.1	Demand of Jobs	64
5.3.2	DBF of Task τ_i	64
5.3.3	DBF of Component $\mathbb{C}$	65
5.3.4	DBF Optimization	67
5.3.5	Schedulability Test and Tolerance Limit	68
5.3.6	Upper Bound of t^{max}	69
5.4	Schedulability Test for Hierarchical Scheduling Framework	71
5.4.1	Execution Strategy under Hierarchical Scheduling	71
5.4.2	MCPR Supply Bound Function	73
5.4.3	Interface Generation	76
5.5	Evaluation	78
5.5.1	Offline Schedulability	79
5.5.2	Online Support for LC Executions	80
5.6	Conclusions and Future Work	83
6	Dynamic Budget Management and Reclamation for High-Criticality Tasks	85
6.1	Motivation and Contribution	87
6.2	Dynamic MC Task and Scheduling Model	88
6.2.1	Task Model	88
6.2.2	EDF-UVD Scheduling Strategy	89
6.3	Maximum Execution-based Budget Allocation (MEBA)	90
6.4	MEBA with Budget Reclamation (MEBA+)	97

TABLE OF CONTENTS

6.4.1	Functions to Refill and Reclaim U-budget	99
6.4.1.1	Budget Refill	99
6.4.1.2	Budget Reclaim	99
6.4.2	Execution Flows of Functions in MEBA+	101
6.4.3	Implementation Complexity of EDF-UVD and MEBA+	103
6.5	Schedulability Test	104
6.5.1	Schedulability Test for MEBA+ and EDF-UVD	109
6.6	Trade-off between α^* and β^*	114
6.6.1	Total System Utilization	115
6.7	Evaluation	118
6.7.1	Minimum Guaranteed Service for LC Tasks	118
6.7.2	Analysis of Probability of Mode-Switch	120
6.7.3	Simulation Results about Mode-Switch	124
6.7.4	Total System Utilization	126
6.8	Conclusions and Future Work	129
7	Conclusions and Future Work	133
7.1	Summary of Contributions	133
7.2	Future Research Directions	136
	References	139

List of Figures

2.1	Calculate execution demand using DBF	12
2.2	Switch from L(m-1) to L(m) mode	15
4.1	Carry-over job $J_{i,k}^*$ has a remaining scheduling window of length x after the system switches to L(m) mode.	32
4.2	Maximum demand of τ_i during $[t^*(m), t^*(m) + t)$	33
4.3	Four types of jobs that experience mode-switch.	34
4.4	$J_{i,A}$: Case $\neg(1) \wedge \neg(4)$	35
4.5	$J_{i,C}$: Case $\neg(1) \wedge \neg(2) \wedge (3)$	36
4.6	$r_{i,A} = -D_i(m-2) + C_i(m-2)$	38
4.7	$r_{i,A} = \text{mod}(t - D_i(m-1), T_i) - T_i$	38
4.8	Case when $t \leq D_i(m-1)$	39
4.9	Case when $t^*(m) \leq C_i(m-1) \wedge t > D_i(m-1)$	39
4.10	Case C3	39
4.11	Case C4	40
4.12	Job release pattern for tasks with $L_i \geq m$	41
4.13	Case when $t - t^*(m) > D_i(m)$	42
4.14	Case when $D_i(m) - D_i(m-1) \leq t - t^*(m) \leq D_i(m)$	43
4.15	Dual-Criticality Task Systems with $RD \in \{0.5, 1\}$ and $P(2) = 0.5$	52
4.16	Dual-Criticality Task Systems with $RD = 0.5$ and $P(2) \in \{0.25, 0.75\}$	53
4.17	Task Systems with Three Criticality Levels	54
4.18	Weighted Acceptance Ratios	55

LIST OF FIGURES

5.1	MCPR worst-case resource supply pattern A	73
5.2	Boundary case for MCPR worst-case resource supply pattern A	73
5.3	MCPR worst-case resource supply pattern B	74
5.4	Schedulability under a Flat Scheduling Framework	80
5.5	Schedulability under a Hierarchical Scheduling Framework	81
5.6	PFJ: $U_\tau = 0.8$	82
5.7	PFJ: $U_\tau \in \{0.8, 0.85, 0.9\}$	83
6.1	Example in which EDF-UVD performs better than EDF-VD	90
6.2	Example illustrating MEBA	91
6.3	Situations to call functions in MEBA	93
6.4	Situations to call functions in MEBA+	102
6.5	Mapping $\mathcal{J}$ to $\mathcal{J}'$	106
6.6	Execution Sequence with Budget Reclamation	110
6.7	Execution Sequence after Updating Execution Requirement	111
6.8	Execution Sequence without Budget Reclamation	113
6.9	β^* and weight w when $SU(w)$ is maximized	116
6.10	Case: $U_H = 0.8$ and $U_L = 0.7$	117
6.11	Average minimum guaranteed service for LC tasks	120
6.12	Average HC service level for LC tasks with varying PH	121
6.13	Comparison of the probability that the mode-switch does not happen	123
6.14	$P(s_i) \sim B(10, 0.4)$ and $P(s_i) \sim B(10, 0.6)$	125
6.15	$U_A = 0.7$, $RC = 3$ and $P(s_i) \sim B(10, 0.4)$	127
6.16	$U_A = 0.7$, $RC = 5$ and $P(s_i) \sim B(10, 0.4)$	128
6.17	$U_A = 0.8$, $RC = 3$ and $P(s_i) \sim B(10, 0.4)$	128
6.18	$U_A = 0.8$, $RC = 5$ and $P(s_i) \sim B(10, 0.4)$	129
6.19	$U_A = 0.7$, $RC = 3$ and $P(s_i) \sim B(10, 0.6)$	129
6.20	$U_A = 0.7$, $RC = 5$ and $P(s_i) \sim B(10, 0.6)$	130
6.21	$U_A = 0.8$, $RC = 3$ and $P(s_i) \sim B(10, 0.6)$	130
6.22	$U_A = 0.8$, $RC = 5$ and $P(s_i) \sim B(10, 0.6)$	131

LIST OF FIGURES

6.23	Total system utilization when $U_L + U_H = 1.1$	131
6.24	Total system utilization when $U_L + U_H = 1.3$	132

List of Tables

1.1	Assurance levels as defined in RTCA DO-178B, a software development process standard used by the United States Federal Aviation Administration (FAA) to certify software in avionics applications.	5
4.1	Notations Used in Chapter 4	28
4.2	Example task set not schedulable under SM DBF-based test	33
5.1	Notations Used in Chapter 5	59
6.1	Notations used in Chapter 6	86
6.2	Mapped tasks in static MC system τ'	105
6.3	Example Task Set	121
6.4	Example distribution for $P(s_i)$	122
6.5	Relative improvement of MBEA+ on number of mode-switches	127

List of Abbreviations

EDF	Earliest-Deadline-First
EDF-VD	Earliest-Deadline-First with Virtual Deadlines
DBF	Demand Bound Function
SBF	Supply Bound Function
MCPR	Mixed-Criticality Periodic Resource
MC	Mixed-Criticality
HC	High-Criticality
LC	Low-Criticality
WCET	Worst-Case Execution Time
EMS	External Mode Switch
IMS	Internal Mode Switch
MEBA	Maximum Execution-based Budget Allocation

List of Publications

- Jaewoo Lee, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, Arvind Easwaran, Insik Shin, and Insup Lee. “Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors,” in Proceedings of the IEEE Real-Time Systems Symposium (RTSS) , 2014, pp. 41-52.
- Xiaozhe Gu, Arvind Easwaran, Kieu-My Phan, and Insik Shin, “Resource efficient isolation mechanisms in mixed-criticality scheduling,” in Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2015, pp. 13-24.
- Xiaozhe Gu and Arvind Easwaran, “Dynamic budget management with service guarantees for mixed- criticality systems,” in Proceedings of the IEEE Real-Time Systems Symposium (RTSS), 2016, pp. 47-56.
- Xiaozhe Gu and Arvind Easwaran, “Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems,” ACM Transactions on Embedded Computing Systems (ACM TECS), vol. 17, no. 1, pp. 24:1-24:24, 2017.
- Xiaozhe Gu and Arvind Easwaran, “Optimal Speed-up Bound for 2-level Mixed-Criticality Arbitrary Deadline Systems”, in Proceedings International Real-Time Scheduling Open Problems Seminar (RTSOPS), 2014, pp.15-16.
- Saravanan Ramanathan, Xiaozhe Gu and Arvind Easwaran, “The Feasibility Analysis of Mixed-Criticality Systems”, in Proceedings International Real-Time Scheduling Open Problems Seminar (RTSOPS), 2016, pp.5-6.

Chapter 1

Introduction

1.1 Real-Time System

The correctness of a real-time system not only depends on the logical result, but also on the time at which the result is produced [1]. A real-time system is said to be predictable if its timing behavior is always within an acceptable range. For example, a patient has a pacemaker inserted in his chest and the pacemaker would provide electrical impulses at regular intervals to help his heart beat. Here the pacemaker must provide service with certain timing constraints, and otherwise it would be fatal. Systems with these kinds of timing constraints are considered as real-time systems.

The timing constraints could appear in different forms, e.g., a period and deadline. Consider a multimedia application which is responsible for compressing video data at regular intervals. The application must finish the current frame before it can process the next one, because otherwise the current frame could be lost. In this system, the period is the frame rate of the system and the deadline is the beginning of the next frame.

Soft real-time systems, for example on-line reservation systems, try to meet deadlines, but if a deadline is missed the penalty is not fatal. On the other hand, hard real-time systems, which are often safety critical, must guarantee that deadlines of all jobs are met and otherwise the consequence could be fatal. For simplicity, when we refer to a real-time system in this dissertation, we presume it is hard real-time.

To guarantee the correctness of a real-time system, its temporal correctness must be validated and verified during the design and implementation process before it can be applied in practice. However, a real-time system is rarely verified directly via exhaustive simulation or testing, because it is costly or even impossible to verify its temporal correctness, and even a simple system could have a large number

of possible execution scenarios. Therefore, formal analysis techniques are necessary to ensure that the designed real-time system is provably correct and predictable.

Typically, there are three steps in the formal analysis process: defining workload models, defining scheduling policy and deriving schedulability test. First, we construct workload models to abstract the system. Such models characterize a system at the proper abstraction level, such that only important information like timing constraints necessary to verify the temporal correctness of the system are included in the model. For example, in the real-time sporadic task model [2], a piece of code that needs to be executed repeatedly is represented by a sporadic task. Each execution instance of this piece of code is regarded as a job of the task. Each task also has a parameter called relative deadline. Suppose a job is released, then absolute deadline of this job is equal to its release time plus the relative deadline, and the job must complete by the absolute deadline. A minimum time interval length between any two successive job releases (defined as the period of this task) is also specified. There are many other different types of such workload models, e.g., periodic [3], multi-frame [4], branching [5, 6], etc.

After the system is abstracted as a workload model, we also need to design a scheduling policy (scheduler) for the system. Given a real-time task system and the platform on which it runs, the scheduling policy would decide which job to be executed on which processor at runtime. For example, the Rate Monotonic (RM) [3] algorithm assigns higher priorities to tasks with shorter periods, and the earliest-deadline-first (EDF) policy dispatches the job with the earliest absolute deadline to the processor for execution during runtime. A good scheduling policy for a real-time task system must have a low runtime complexity for making scheduling decisions, and must be able to schedule a large proportion of feasible task systems¹

The scheduling policies can be broadly classified into two types:

1. Static Scheduling: In static scheduling, the scheduling decisions are made off-line. A runtime schedule is generated off-line based on the prior knowledge of system parameters. Given a set of jobs, static scheduling aims to construct a schedule (template or calendar) in which each job meets its deadline (a feasible schedule).
2. Dynamic scheduling: On the other hand, dynamic scheduling is flexible and adaptive, and it determines at runtime which job to run next. Dynamic scheduling policies can be further categorized into fixed-priority ones and dynamic-priority ones.

(a) Fixed-Priority (e.g., RM): priorities are assigned to individual tasks, i.e., jobs of the same

¹If a task system is feasible, it must be schedulable by some clairvoyant or non-clairvoyant scheduling policy.

task share the same priority.

- (b) Dynamic-Priority (e.g., EDF): priorities are assigned to individual jobs. For example, the priority of any job under EDF depends on its absolute deadline.

Given a system and a certain scheduling policy, a schedulability test should be able to determine whether the system is schedulable by the scheduling policy or not. Thus, a schedulability test for a certain scheduling policy and workload model must be derived to validate the temporal correctness of the system at design time. While a necessary schedulability test only provides necessary conditions for a task set to be schedulable, an exact schedulability test is both necessary and sufficient. Even though schedulability tests with low complexity are preferred, time-consuming tests with high complexity are still acceptable as long as they can significantly improve the runtime performance and resource utilization, because these tests are only used off-line.

1.1.1 Hardware platforms

The computing platform upon which a system is running is important in modeling a real-time system, because the execution time of a given piece of code is associated with a certain platform. Typically, a processor is assumed to run at a fixed speed of one in real-time systems research.

While a uniprocessor platform only has one processor, a multiprocessor platform is a combination of multiple uniprocessors. A multiprocessor platform could be an identical multiprocessor if all the processors on it run at the same speed, or a uniform multiprocessor if each processor has its own execution speed. In this dissertation, we consider a uniprocessor platform that runs at a fixed speed of one.

1.2 Mixed-Criticality System (MCS)

In embedded systems, there is a trend in which multiple functionalities with different levels of “criticality” (or importance) are developed independently and integrated together on a common computing platform [7]. This trend is evident in industry-driven initiatives such as ARINC653 [8], Integrated Modular Avionics (IMA) in aviation and AUTOSAR [9] in automotive.

Criticality represents the level of assurance against severe failure needed for a system. Typically, failure of functionality with higher level of criticality would cause a more severe consequence to the whole system than less critical functionalities. For simplicity of analysis, many works on scheduling of mixed-criticality (MC) systems are restricted to two criticality levels, i.e., high-criticality (HC) and

low-criticality (LC), and such systems are referred to as dual-criticality systems. In a dual-criticality system, a task would either be an LC task or an HC task.

However, in practice, a system may have upto five criticality levels. For example, see IEC 61508, DO-178B, DO-254 and ISO 26262 standards [10–13].

Table 1.1 lists the criticality levels from RTCA DO-178B avionics software standard [11]. The RTCA DO-178B standard divides tasks into five assurance levels, from level A to level E, where level A denotes the highest assurance level and level E denotes the lowest. Failure of a level A task will have catastrophic results, while failure of a level E task will not cause any safety-critical issues.

Table 1.1: Assurance levels as defined in RTCA DO-178B, a software development process standard used by the United States Federal Aviation Administration (FAA) to certify software in avionics applications.

Level	Failure Condition	Interpretation.
A	Catastrophic	Failure may cause a crash.
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.
C	Major	Failure is significant, but has a lesser impact than a Hazardous failure.
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure.
E	No effect	Failure has no impact on safety, aircraft operation.

The challenge in such systems is the dramatically-different nature of less critical functions characterized by best-effort or soft real-time needs and the more critical control functions that require strong reliability. To balance the conflicting requirements, i.e., isolate the hard real-time applications for safety assurance and meanwhile enable efficient resource sharing for economic benefits, the concept of *mixed-criticality* (MC) appears to be important.

1.2.1 Mixed-Criticality Scheduling

The fundamental research question underlying mixed-criticality systems is: *How to balance the conflicting requirements of isolation for safety assurance and efficient resource sharing for economic benefits due to size, weight, and power (SWaP) constraints.*

In order to certify a system as being correct (i.e, providing safety assurance), safety-critical functions are conventionally verified under extremely rigorous and pessimistic assumptions, which are unlikely

to occur in practice. One task parameter upon which such assumptions are generally imposed is the Worst-Case Execution Time (WCET) of a task. For a specific task and a particular platform upon which it is to execute, the WCET denotes an upper bound on the amount of time the task takes to execute upon that platform. The value of WCET can never be exceeded by the actual execution time of any instance of this task at runtime, otherwise the real-time scheduling techniques cannot guarantee the correctness of the system. However, determining an exact WCET value for a task is very difficult and remains an active area of research. As a result, the WCET parameter used by the certification authorities is typically a very conservative upper bound that highly exceeds (sometimes can be several times as much as) the actual WCET.

Such pessimistic assumptions in WCET estimations of tasks can guarantee that the resources reserved for critical functionalities are always sufficient. Though, from a certification perspective, the system can be designed to be safe by making such assumptions, the system resources are severely under-utilized in practice. On the other hand, it is not reasonable to require that all deadlines, including those of less critical tasks, are always met, because an occasional deadline miss of a less critical task would not have any severe consequence.

In practice, the probability that a task executes as much as its actual WCET is very low. A reasonable low-confidence WCET estimation, even if it is based on measurements, may be sufficient for almost all possible execution scenarios of this task. Motivated by this observation, Vestal [14] proposed a multi-WCET mixed-criticality task and scheduling model (Vestal's model), and formally defined the scheduling requirements of such mixed-criticality systems.

In this model, the correctness of functionalities at different criticality levels are guaranteed with different degrees of confidence. At runtime, if a task executes beyond its low-confidence WCET estimation without signaling completion, it could trigger an execution mode-switch to the whole system, after which all tasks with lower criticality level are no longer required to meet deadlines. The resource reserved for those tasks with lower criticality level could be then consumed by the more critical tasks to ensure their correctness. As long as the low-confidence WCET estimations satisfy the execution requirement of most execution scenarios, we can expect that all deadlines of different criticality levels would be met most of the time.

1.3 Motivation and Problem Statement

Based on the abstract mixed-criticality workload model proposed by Vestal [14], several studies on mixed-criticality systems have focused on the design of scheduling policies and schedulability tests for uniprocessors (e.g., see [15] for a detailed review). Due to the fact that mixed-criticality scheduling is NP-hard in the strong sense even when there are only two criticality levels [16], these studies have developed efficient approximation algorithms that aim to improve schedulability. Although these algorithms have very good performance for systems with two criticality levels, their performance does not scale well with the number of criticality levels. *In particular, for dynamic-priority algorithms such as those based on EDF, there is a significant drop in algorithm performance as the number of criticality level increases.* **Thus, there is a need to design algorithms and tests whose performance scales well with the number of criticality levels, while still maintaining an efficient test complexity.** In this dissertation, we propose an improved analysis technique and EDF-based scheduling strategy to address this scalability problem.

Although Vestal’s mixed-criticality workload model has abstracted some key properties of mixed-criticality systems, it has received several criticisms that it makes unrealistic assumptions [17]. In particular, system engineers have complained that the model does not meet their expectations, and hence the studies based on those models are unlikely to be used in practice [18]. Some of these critical issues are presented below.

- Low-criticality tasks cannot be completely abandoned after an execution mode-switch, and should instead execute with degraded service as long as they do not affect the correctness of more critical tasks.
- The system should be able to return to the normal execution mode with full service for all the tasks as soon as possible.
- It is not reasonable to trigger an execution mode-switch and immediately suspend or degrade service to all the low-criticality tasks once a high-criticality task exceeds its low-confidence WCET estimation.
- The WCET estimations in this model are primarily used as “budgets” for tasks. However, it is an undue burden on the application designer to derive these multiple WCET estimations for each task for this purpose.

Addressing the above issues can therefore be regarded as some of the most important challenges in mixed-criticality real-time scheduling. Realizing this, the research community has been active in

providing support for the execution of low-criticality tasks using various techniques and approaches (e.g., see [19–22]). In spite of these advances in mixed-criticality models and/or scheduling strategies, a couple of key questions remain unanswered.

1. **Is there a reasonable strategy to offer differentiated service to the low-criticality tasks depending on the specific high-criticality tasks that exceed their low-confidence WCET estimations?** That is, instead of penalizing all the low-criticality tasks to the same extent, would it be possible to vary the penalty depending on the specific runtime scenario? Providing such a differentiated service has practical relevance, because these tasks (i.e., functions) generally have some dependencies with each other.
2. **Since low-confidence WCET estimations are only used as task budgets to determine when an execution mode-switch must be triggered, would it be possible to derive these budgets based on schedulability considerations instead of using WCET estimations?** This will not only relieve the application designer from having to derive these WCET estimations, but may also improve the support for low-criticality tasks if budgets are carefully assigned.

In this dissertation, we propose novel mixed-criticality task models and EDF-based scheduling strategies that address the above questions.

1.4 Contributions and Organization

The main technical contributions and organization of this dissertation are presented below:

- In Chapter 2, we introduce some results in traditional real-time scheduling theory that are relevant for this dissertation (e.g., sporadic real-time model, EDF scheduling policy). Then, we present Vestal’s mixed-criticality task and scheduling model and its execution semantics in detail.
- In Chapter 3, we present related work. We provide an overview of the main results in mixed-criticality scheduling theory focusing on uniprocessors. We also discuss approaches and strategies that have been developed to support the execution of low-criticality tasks.
- In Chapter 4, we present our results on improving the performance scalability of EDF-based scheduling with the number of criticality levels in the system. We first introduce the existing demand bound function (DBF) based schedulability tests [23–25], and discuss their limitations.

Then we propose a new DBF-based schedulability test and corresponding EDF-based scheduling strategy for systems with multiple criticality levels. The complexity of this test is independent of the number of criticality levels, but nevertheless its performance scales significantly better than the existing tests for EDF-based strategies.

- In Chapter 5, we present our results on providing differentiated service to the low-criticality tasks depending on the specific high-criticality tasks that exceed their low-confidence WCET estimations. We propose a component-based model for mixed-criticality systems. This model comprises a component-level design parameter called tolerance limit, which represents the expected number of high-criticality tasks within the component that will require additional resources at the same time. An execution strategy based on this parameter is also proposed, such that as long as WCET exceedance events are within the tolerance limit, low-criticality tasks in other components remain unaffected. Thus, to achieve strong temporal and spatial partitioning [26], the component boundaries of the proposed model are designed to provide the isolation necessary to support the execution of low-criticality tasks, and at the same time guarantee the correctness of the high-criticality ones.
- In Chapter 6, we present our results on deriving high-criticality task budgets based on schedulability considerations instead of using low-confidence WCET estimations. We propose a *dynamic mixed-criticality task and scheduling model* in which high-criticality task budgets are determined at runtime depending on the execution scenario. To ensure a safe execution for all the high-criticality tasks, resources are always guaranteed to them upto their WCETs, a single value provided by the application designer. Off-line, we use schedulability analysis to determine a *total budget allocation* (single value) for all the high-criticality tasks combined. Execution mode-switch occurs in this model only when the resource utilization of all the high-criticality tasks collectively exceed this total budget. Then, at runtime, we use a strategy to allocate budgets to individual high-criticality tasks, with the objective of postponing the execution mode-switch as much as possible. Finally, based on the observation that a task may terminate before it uses up its allocated budget, we also propose a budget reclamation scheme that can work along with the dynamic model to further support the execution of low-criticality tasks.
- In Chapter 7, we summarize the dissertation and discuss potential directions for future research work.

Chapter 2

Background

In this chapter, some important results in non-MC real-time scheduling that are relevant for this dissertation are introduced, e.g., the sporadic task model [2] and the Earliest-deadline-first (EDF) scheduling policy. Then, we present the details of Vestal’s MC task and scheduling model and its execution semantics [14].

2.1 Real-Time Scheduling Theory

2.1.1 Sporadic Task Model

The classic sporadic task model [2] is a generic model for capturing the real-time requirements of many event-driven systems including those with MC such as avionics and automotive. The principle of the classic real-time task model is that a piece of code becomes available for execution at a time instant, which is treated as a new job of this task, and takes a certain amount of time to finish execution, and must complete execution by its absolute deadline.

A sporadic task [2] can be specified as $\tau_i = (T_i, C_i, D_i)$, where T_i denotes minimum separation between successive job releases (defined as the task’s period), D_i denotes its relative deadline, and C_i denotes the Worst-Case Execution Time (WCET) of τ_i . For a specific piece of code and the platform upon which it is to execute, the WCET of the code denotes an upper bound on the amount of time the code takes to execute upon this platform. The utilization of task τ_i is defined as

$$u_i = \frac{C_i}{T_i}, \quad (2.1)$$

i.e., fraction between C_i and T_i . Given a task set $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$, the utilization of τ is defined as

$$U = \sum_{\tau_i \in \tau} u_i \quad (2.2)$$

Definition 1 (Schedulable) *A sporadic task set τ is said to be schedulable by a certain scheduling policy if all tasks can execute upto C_i time units and signal completion between their release times and deadlines.*

According to the relationship between T_i and D_i , a sporadic task can be categorized into three cases. Task τ_i is an *implicit-deadline* task if its deadline D_i is equal to its period T_i . If the deadline D_i of τ_i is no greater than its period T_i , then τ_i is a *constrained-deadline* task. Finally, if τ_i does not have any restriction on the relation between its period T_i and deadline D_i , then τ_i is an *arbitrary-deadline* task.

2.1.2 Analysis for Earliest-deadline-first (EDF) Scheduling

In this dissertation, we focus on EDF-based scheduling policy for MC systems. Therefore, in this section, we first formally define EDF scheduling policy. Then we introduce the utilization based and demand bound function (DBF) based schedulability tests [27] for EDF scheduling of non-MC systems.

Definition 2 *Earliest-deadline-first (EDF) scheduling policy always schedules the job with the earliest absolute deadline among all available jobs at any time instant.*

EDF has an efficient utilization based schedulability test for implicit-deadline sporadic task sets as shown in Theorem 1.

Theorem 1 ([3]) *A real-time implicit-deadline sporadic task set is schedulable by EDF scheduling policy on unit-speed uniprocessor platforms if and only if its utilization is no greater than 1.*

To analyze the schedulability of EDF for more general real-time workloads [27], the DBF-based schedulability test is proposed. DBF captures the maximum execution demand a task can generate for a given time interval length.

Definition 3 (Demand bound function) *The demand bound function $dbf(\tau_i, t)$ gives an upper bound on the maximum possible execution demand of task τ_i in any time interval of length t , where demand is calculated as the total amount of required execution time of jobs with their whole scheduling windows within the time interval.*

For example a task $\tau_1 = (T_1 = 5, C_1 = 2, D_1 = 3)$ is supposed to generate at most $2 \times C_1 = 2 \times 2 = 4$ units of demand in a time interval length equal to 8 as shown in Figure 2.1. The demand bound function

Figure 2.1: Calculate execution demand using DBF

for a time interval length t for a standard sporadic task can be computed in constant time [27]. As long as we can guarantee that the total demand bound function of a task set τ is always smaller than or equal to the time interval length t , we can claim τ is schedulable by EDF algorithm on a dedicated unit-speed uniprocessor platform.

Theorem 2 ([27]) *A sporadic constrained deadline task set τ with utilization less than 1 is successfully scheduled by the earliest deadline first (EDF) algorithm on a dedicated unit-speed uniprocessor platform if*

$$\forall 0 < t \leq t^{max} : \sum_{\tau_i \in \tau} dbf(\tau_i, t) \leq t.$$

Here

$$t^{max} = \min \left\{ lcm + \max_{\tau_i \in \tau} \{D_i\}, \frac{U}{1 - U} \max_{\tau_i \in \tau} \{T_i - D_i\} \right\},$$

where lcm is the least common multiple of periods of tasks in τ .

2.2 MC Task and Scheduling Model

In this section, we present Vestal's MC task and scheduling model as well as its execution semantics [14]. Let $M \in \mathbb{Z}^+$ denote the number of criticality levels in a MC system. Each task is assumed to be assigned a criticality level $L_i \in \{1, 2, \dots, M\}$, expressing its degree of importance, with a larger value denoting a greater level of importance. MC real-time systems need to pass safety certification, though the deadlines of less critical applications may be missed occasionally when critical applications overload because they do not influence the safety of the system. Vestal [14] proposed a new model to abstract MC systems and also defined the scheduling requirements for such models. Under Vestal's model, different levels of correctness are guaranteed under different execution modes during runtime.

The execution mode of a system depends on the resource requirements of jobs; it changes when jobs with higher criticality levels demand additional resources.

2.2.1 Vestal's MC Model

Vestal's MC task model [14] is widely used in many previous works on scheduling of MC systems, e.g., [14, 23, 28–30]. This model is a simple extension of the classic sporadic task model [2] to an MC setting except that; the WCET estimations of a single task can vary between criticality levels. Thus, in general, a sporadic MC task τ_i in this model can be defined as follows.

Definition 4 *A sporadic MC task τ_i can be represented as a 4-tuple of parameters: $\tau_i = (T_i, \vec{C}_i, D_i, L_i)$, where:*

- $T_i \in \mathbb{N}_+$ *is the minimal time separation between the release of two successive jobs which is also called period;*
- $L_i \in \mathbb{N}_+$ *is the criticality level of τ_i ($L_i = 1$ denotes the lowest criticality level);*
- $D_i \in \mathbb{Q}_+$ *is the relative deadline;*
- $\vec{C}_i = (C_i(1), C_i(2), \dots, C_i(L_i))$ *is a vector of WCET estimations, where $C_i(k) \in \mathbb{Q}_+$ specifies the WCET estimation of τ_i when the system is in level k criticality mode.*

Since the WCET estimations for higher criticality levels are measured more conservatively, the model makes the assumption that $C_i(k)$ is monotonically non-decreasing as the value of k increases, i.e.,

$$\forall \tau_i \in \tau : C_i(1) \leq C_i(2) \leq \dots \leq C_i(L_i) \leq D_i \leq T_i.$$

This is reasonable because larger values of k correspond to greater degrees of confidence, and hence are likely to have larger WCET estimations. The utilization of an MC task and an MC task set can be defined as follows.

Definition 5 *Given an MC sporadic task set τ , the utilization of task τ_i at criticality level k is defined as*

$$u_i(k) = \frac{C_i(k)}{T_i} \tag{2.3}$$

Similarly, the utilization of τ at criticality level k is defined as

$$U(k) = \sum_{L_i \geq k} u_i(k) \quad (2.4)$$

Note that the utilization of τ at a specific criticality level k only considers tasks with criticality level at least k .

2.2.2 Dual-Criticality Model

Dual-criticality model is a special case of the general multi-criticality model, in which a task can either be a low-criticality (LC) task or a high-criticality (HC) task. Most existing works on MC scheduling [16, 19–21, 24, 31] focus on just two criticality levels to simplify the analysis process.

Typically, the criticality level of τ_i in a dual-criticality model is denoted as $L_i \in \{LC, HC\}$ instead of $L_i \in \{1, 2\}$, and its WCET estimations for LC and HC criticality levels is denoted as $\vec{C}_i = (C_i(L), C_i(H))$ instead of $\vec{C}_i = (C_i(1), C_i(2))$. Analogously, in a dual-criticality model, we use $u_i(x)$ and $U(x)$ ($x \in \{L, H\}$) to denote task utilization and system utilization, respectively. In Chapters 5 and 6 we consider a dual-criticality system, whereas in Chapter 4 we consider a multi-criticality system.

2.2.3 Execution Semantics of Vestal's Model

In this section, we introduce the execution semantics of Vestal's model, i.e., when an execution mode-switch should be triggered and less critical tasks should be deprioritized. The system initially starts in L(1) (short for level one) mode (i.e., the lowest criticality mode), and as long as no job of task τ_i executes beyond $C_i(1)$ time units, it remains there. Any task $\tau_i \in \tau$ can give rise to a potentially unbounded sequence of jobs $\langle J_{i,1}, J_{i,2}, \dots \rangle$ in the standard way for traditional real-time sporadic tasks, where $J_{i,k}$ denotes the k_{th} job of τ_i . In L(1) mode, the deadline of each job is required to be met.

Once any job executes for its WCET estimation for L(1) mode, i.e., $C_i(1)$ time units, but does not signal completion, the execution mode of the system immediately switches to L(2). However, in L(2) mode, only deadlines of jobs with $L_i \geq 2$ are required to be met.

Intuitively, all jobs are expected to exhibit their L(1) criticality behavior, i.e., execute for less than $C_i(1)$ units, for most of the time. On the other hand, even if some job exhibits L(2) criticality behavior by requesting additional resource, all jobs with $L_i \geq 2$ can still complete successfully, while jobs with

$L_i < 2$ may be ignored.

Figure 2.2: Switch from $L(m-1)$ to $L(m)$ mode

Analogously, suppose the system is currently in $L(m-1)$ mode which indicates at least one job has executed beyond $C_i(m-2)$ but no more than $C_i(m-1)$, then all deadlines of jobs with $L_i \geq m-1$ are required to be met. As shown in Figure 2.2, the system would stay in $L(m-1)$ mode as long as no job executes beyond $C_i(m-1)$. When some job first executes for $C_i(m-1)$ time units but does not signal that it has finished, the system switches to $L(m)$ criticality mode. The time instant when an execution mode-switch is triggered is called mode-switch instant and is defined as follows.

Definition 6 (Mode-Switch Instant) *The time instant when the system switches from $L(m-1)$ to $L(m)$ mode is called mode-switch instant for $L(m)$ mode and is denoted as $t^*(m)$.*

After $t^*(m)$, deadlines of jobs with $L_i < m$, including those that are released before $t^*(m)$ and have deadline after $t^*(m)$, are no longer required to be met. However, all the deadlines of jobs with $L_i \geq m$ must still be met, even if they execute upto $C_i(m)$ time units. For simplicity, many existing MC scheduling studies based on this model assume that all the jobs with $L_i < m$ are discarded after $t^*(m)$.

The execution mode of an MC system at any time instant t can then be defined as follows.

Definition 7 *At some arbitrary time instant t , an MC system τ is in $L(m)$ mode if m is the smallest integer such that no job released before t has executed for more than its WCET estimation for $L(m)$ mode, i.e., $C_i(m)$, by t .*

Since Vestal's model only guarantees the correctness of tasks with different criticality levels under different execution modes, an MC system is *MC-schedulable* by a certain scheduling policy if the

following conditions are satisfied.

Definition 8 (MC-Schedulable) *An MC task system is MC-schedulable by a certain scheduling policy if $\forall m : 1 \leq m \leq M$ where $M = \max_{\tau_i \in \tau} \{L_i\}$, all jobs with criticality level $L_i \geq m$ can execute upto $C_i(m)$ time units and signal completion between their release times and deadlines while the system stays in $L(m)$ mode.*

Job level execution semantics: We now consider the behavior at the job level. An MC job $J_{i,k}$ is released at $r_{i,k}$, has an execution requirement of $E_{i,k}$ within its time window $[r_{i,k}, d_{i,k}]$ where $d_{i,k} = r_{i,k} + D_i$, and has executed for $e_{i,k}$ time units within $[r_{i,k}, d_{i,k}]$. Note that the value of $E_{i,k} (\leq C_i(L_i))$ is not known beforehand, and is revealed when $J_{i,k}$ signals that it has completed execution. If $J_{i,k}$ has executed for less than $E_{i,k}$ time units by its deadline, then $J_{i,k}$ misses its deadline. Any deadline miss is regarded as system failure. Thus, if $J_{i,k}$ misses its deadline, $E_{i,k} > e_{i,k}$ must hold.

However, $J_{i,k}$ may be ignored after an execution mode-switch, and thereafter the deadline of $J_{i,k}$ is no longer required to be met. In this case, we assume its execution requirement $E_{i,k}$ is equal to its consumed budget, i.e., $e_{i,k}$. Thus, $J_{i,k}$ would not be regarded as missing its deadline since its deadline is not required to be met.

2.2.3.1 Budget and WCET Estimation

From the description of the execution semantics of Vestal's model in the above section, we can observe that the WCET estimation $C_i(m)$ is primarily used as a “budget” for task τ_i in the $L(m)$ mode. It also helps the scheduler to determine whether the system should switch its execution mode to a higher level and deprioritize tasks with lower criticality levels.

Consider the following definition of execution mode budget for an MC task.

Definition 9 $B_i(m)$ denotes the maximum budget allocated to jobs of τ_i by an MC scheduler when the system is in the $L(m)$ mode.

As we can observe, under Vestal's model, the execution semantics assumes a static budget allocation in each mode, i.e.,

$$B_i(m) = C_i(m). \quad (2.5)$$

Thus, in Vestal's model, each task is required to have a fixed (static) budget allocation in each execution

mode. For the dual-criticality case, the budget allocated to τ_i in LC mode (LC budget) must be $B_i(L) = C_i(L)$, and the budget allocated to τ_i in HC mode (HC budget) must be $B_i(L) = C_i(H)$.

2.3 Brief Introduction of EDF-VD*

In this dissertation, we consider the EDF-VD*¹ scheduling policy (a variant of EDF scheduling policy). Hence, in this section, we first provide a brief introduction about it.

Suppose an MC system switches from L(m-1) to L(m) mode at $t^*(m)$. Then job $J_{i,k}$ is a *carry-over* job if $t^*(m)$ is within its scheduling window.

Definition 10 (Carry-Over Job) Suppose an MC system switches from L(m-1) to L(m) mode at $t^*(m)$. A job $J_{i,k}^*$ ($L_i \geq m$) is called a carry-over job if $r_{i,k} \leq t^*(m)$ and $r_{i,k} + D_i > t^*(m)$.

The original EDF policy is not very efficient for scheduling MC systems, because EDF only prioritizes jobs with urgent deadlines irrespective of their criticality level. As a result, it may execute a carry-over job quite late and hence the carry-over job does not have enough time to finish its additional execution demand, i.e., $C_i(m) - C_i(m-1)$, by its deadline. By assigning (*shorter*) *virtual deadlines* [23, 24, 33] to tasks in different execution modes, the EDF scheduler could find a balance between prioritizing jobs with earlier deadlines and jobs with higher criticality levels. Thus, task τ_i scheduled by EDF-VD* has a list of virtual deadlines for different execution modes, i.e.,

$$\vec{D}_i = (D_i(1), D_i(2), \dots, D_i(L_i) = D_i)$$

The absolute virtual deadline of job $J_{i,j}$ for L(m) mode can be denoted as $d_{i,j}(m) = r_{i,j} + D_i(m)$, where $m \in \{1, 2, \dots\}$. Similarly, for a dual-criticality system, $d_{i,j}(L) = r_{i,j} + D_i(L)$ denotes the absolute virtual deadline of $J_{i,j}$ in the LC mode.

RunTime Operation of EDF-VD*: Assuming the virtual deadlines of all tasks are already determined off-line, the runtime operation of EDF-VD* can be summarized as follows. Suppose job $J_{i,k}$ arrives at $r_{i,k}$ and the system is in L(m-1) mode at this time. Then this job is assigned a virtual deadline equal to $d_{i,k}(m-1) = r_{i,k} + D_i(m-1)$. If some job executes beyond its WCET estimation for L(m-1) mode, i.e., $C_i(m-1)$, without signaling that it has completed execution, the following changes occur:

¹We use EDF-VD* to denote any EDF scheduler that applies virtual deadline scheme in order to avoid confusion with the EDF-VD algorithm [32, 33].

1. All currently-active jobs with $L_i < m$ are immediately discarded; henceforth, no job with $L_i < m$ will receive any execution.
2. Subsequent runtime scheduling of tasks (including their jobs that are currently active) will switch to the virtual deadline for L(m) mode, i.e., $D_i(m)$.

Thus, by using virtual deadlines, carry-over job $J_{i,k}^*$ would have at least $D_i(m) - D_i(m-1)$ time units to meet its additional execution demand of $C_i(m) - C_i(m-1)$ after $t^*(m)$, albeit at the cost of a higher load of execution demand in L(m-1) mode.

Chapter 3

Related Work

Since Vestal first proposed the mixed-criticality (MC) task and scheduling model [14], extensive research has been done on MC scheduling under this model. An up-to-date review on MC scheduling can be found in [15]. In this chapter, we introduce some important results in MC scheduling theory that are related to the dissertation. We focus on scheduling policies and schedulability tests for uniprocessor platforms as well as strategies to support the execution of low-criticality (LC) tasks.

3.1 Scheduling Policies and Schedulability Tests

Most studies on the scheduling of MC systems can be categorized into 3 types: job scheduling, fixed-priority scheduling and EDF scheduling. Initial studies addressed the problem of scheduling a finite set of MC jobs with varying WCET estimations for different criticality modes on a uniprocessor platform [16, 28, 34, 35]. However, in many practical real-time systems, a task is able to generate an infinite number of jobs. Thus, works on scheduling MC jobs are superseded by studies that can be applicable to the more general sporadic MC task system.

3.1.1 Fixed-Priority Scheduling

In fixed-priority MC scheduling, Response Time Analysis (RTA) technique is usually employed to determine the schedulability of an MC system. RTA can be used to get the worst case completion time of a task τ_i (denoted as R_i). We can compare R_i with the relative deadline D_i to check whether τ_i would miss its deadline. Given a certain priority assignment, if $\forall \tau_i \in \tau : R_i \leq D_i$, then we can claim τ is

schedulable.

Vestal's approach [14] is the first work that uses RTA to analyze the schedulability of an MC system. It applies Audsley's priority assignment algorithm [36] to assign a priority for each task. In [37], Dorin et al. extended it to include release jitter, and present a sensitivity analysis for multiple criticality tasks. Although Vestal's approach allows the priorities of tasks with different criticality levels to be interleaved, it allows task τ_i to execute beyond $C_i(L_i)$ time units. As a result, the platform is under-utilized.

The above issue on platform under-utilization was later improved by the Static Mixed Criticality Scheme (SMC) [38], which does not allow a job to execute for more than $C_i(L_i)$ time units. SMC improves resource usage by monitoring the task executions, and preventing the execution from over-running its allocated budget. When formalizing the response time of τ_i , SMC considers three different cases depending on whether a higher priority task τ_j has an equal, higher or lower criticality than a task τ_i under analysis. If $L_i = L_j$ then normal representative value $C_i(L_j)$ can be used. If $L_i \leq L_j$ then we do not need to consider the scenario when τ_j executes beyond $C_j(L_i)$ because by that time τ_i has already been dropped. Finally, if $L_i > L_j$ then we just need to ensure τ_j does not execute for more than $C_j(L_j)$ time units. In sum, the response time of τ_i can be computed as follows.

$$R_i = C_i(L_i) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\min\{L_i, L_j\}) \quad (3.1)$$

Here $hp(i)$ denotes the subset of tasks with higher priority than τ_i .

Adaptive Mixed Criticality (AMC) scheduling [16] (constrained to dual-criticality system) goes further and does not allow LC jobs to execute at all if any high-criticality (HC) job executes for more than $C_i(L)$ time units. This work exhibits that it dominates all the previous fixed-priority MC scheduling works in terms of being able to schedule a wide variety of task systems. However, it is constrained to dual-criticality systems.

AMC analyzes the schedulability of LC mode, HC mode and transition (LC to HC) mode, separately. For task τ_i , its response time for LC mode $R_i(L)$ is computed first. If task τ_i is impacted by a mode-switch change, it must occur before $R_i(L)$, because otherwise it must have already completed by that time instant. When computing the response time of τ_i in transition mode (denoted as R_i^*), all LC tasks are assumed to be abandoned by the mode-switch time instant t^* . AMC comprises two methods for computing R_i^* . AMC-rtb considers a single upper bound of t^* , i.e., $R_i(L)$. AMC-max looks at all the possible values of t^* before $R_i(L)$ and chooses the largest R_i^* .

AMC is extended to an arbitrary number of criticality levels by Fleming and Burns [39]. It focuses on

five levels as it is the maximum number of criticality levels found in automotive and avionics standards. As found in [39], the schedulability performance of AMC-rtb is similar to AMC-max as the number of criticality levels increases. Since AMC-max is computationally expensive, it is concluded that AMC-rtb is an adequate and effective substitute for AMC-max. In [40], Zhao et al. extended the AMC-rtb approach by incorporating preemption thresholds into the model. In [41], Burns and Davis applied deferred preemption to fixed-priority MC scheduling, where each task has a final non-preemptive region. They also showed that AMC with deferred preemption has better performance than AMC-rtb in terms of acceptance ratios¹. In AMC (including variants of AMC) and SMC, the priority of tasks would not change after a criticality mode change. A simple form of sufficient analysis for PMC (priority may change) is proposed in [42], but evaluation results show that PMC and AMC-rtb do not dominate each other.

3.1.2 EDF Scheduling

Baruah and Vestal [43] first considered MC scheduling with EDF. They consider a hybrid scheduling policy, where tasks do not necessarily have a unique priority. As long as jobs in one priority are awaiting execution during runtime, no lower-priority jobs may execute. Within each priority, tasks will be scheduled using EDF. Subsequently, Park and Kim [44] proposed Criticality Based EDF that runs HC jobs as late as possible without missing deadlines. On the other hand, the LC jobs would only run in the slack generated by HC jobs. Meanwhile, tasks with the same criticality level are scheduled by EDF.

A more general demand bound function based analysis for EDF MC (dual-criticality) scheduling was proposed by Ekberg and Yi [24]. They assign a virtual deadline $D_i(L) (< D_i)$ to HC tasks. At runtime, all HC tasks are scheduled by EDF according to their virtual deadlines in LC mode. After the system switches to HC mode, all LC tasks would be dropped and the HC tasks would revert to their actual deadlines. This scheme (virtual deadline) can increase the likelihood that less urgent HC jobs execute earlier than more urgent LC jobs. It also enables the HC jobs to have at least $D_i - D_i(L)$ units of additional time to finish the additional execution demand of $C_i(H) - C_i(L)$. They also introduce a heuristic deadline tuning algorithm to increase the chances that a task set is MC-schedulable by the proposed test. Later, their work [25] extended this analysis to a more generalized model with more than two criticality levels. *However, these techniques are not good at scheduling task systems with a larger percentage of HC tasks and their performance drops dramatically as the number of criticality levels increases.* A tighter DBF-based schedulability analysis and corresponding virtual deadline assignment

¹Acceptance ratio for a schedulability test A denotes the proportion of the number of schedulable tasks as determined by A to the total number of generated task sets.

strategy was proposed by Easwaran [23], but it is restricted to dual-criticality systems.

Baruah et al. [32] proposed a similar strategy for dual-criticality MC scheduling named EDF-VD (earliest deadline first-virtual deadline). Unlike in [24], where different HC tasks are assigned virtual deadlines with a flexible scheme, EDF-VD uses a single scaling factor to set virtual deadlines for all HC tasks, i.e., $\forall \tau_i : D_i(L) = \alpha_i \times D_i$, where $\alpha_i \in [0, 1]$. Later, they extended EDF-VD to multi-criticality arbitrary deadline systems in [33]. Though the utilization based schedulability analysis for EDF-VD has lower acceptance ratios than the DBF-based analysis [23–25] for a variety of task systems, it has lower complexity and also provides a speedup factor². Masrur et al. proposed an intermediate solution [45], where HC tasks that experience a small increase of workload are assigned one deadline scaling factor, whereas tasks with a large increase of workload are assigned a separate deadline scaling factor.

As discussed in Section 1.3 in Chapter 1, these existing studies for EDF-based scheduling of MC systems are either not applicable for multi-criticality systems (e.g., [23, 24, 32]), or their performance in terms of acceptance ratios for a variety of MC task systems scales very poorly with the increasing number of criticality levels (e.g., [25, 33]). The only known DBF-based test for multi-criticality systems also does not show good performance when the proportion of HC tasks in the system increases [25]. To overcome these limitations in multi-criticality system scheduling, we derive a new DBF-based schedulability test and corresponding virtual deadline assignment strategy for EDF MC scheduling in Chapter 4. This test effectively combines the two techniques proposed in [24] and [23], thus achieving very good performance even for increasing number of criticality levels, without any significant increase in the test complexity.

3.2 Support for the Execution of LC Tasks

Most existing studies in MC scheduling (e.g., see [16, 23–25, 32, 33]) would under-utilize the platform, because they would penalize all LC tasks in order to ensure that all HC tasks are guaranteed sufficient resources. For simplicity, these studies assume that all LC tasks are abandoned once the system switches to HC mode. They further assume that all HC tasks in the system would thereafter request additional resources, upto their WCET estimations for HC mode $C_i(H)$.

As pointed out in [17], when a single HC task exceeds its WCET estimation for LC mode, the likelihood that all the other HC tasks in the system will also request additional resources simultaneously

²The speedup factor for a scheduling policy A is defined as the minimum factor by which the speed of the processor would need to be increased such that all systems that are schedulable according to an optimal (possibly clairvoyant) scheduling policy become schedulable under policy A.

is low in practice. Thus, it is not reasonable to penalize all LC tasks when this event occurs. As a result, lots of strategies are proposed to support the execution of LC tasks as much as possible, while still ensuring that all HC tasks can always receive sufficient resources.

These strategies can be broadly classified into four categories:

1. The first category of studies, e.g., [19–22], support LC tasks by offering a degraded (and in some cases guaranteed) service to all of them after the system switches to HC mode. Instead of being dropped, these strategies would either reduce the dispatch frequency of jobs or reduce the budget allocated to LC jobs when the system is in HC mode.
2. The second category of studies, e.g., [46,47], support LC tasks by limiting the impact of a certain HC task exceeding its WCET estimation for LC mode. In the models proposed in these works, only a subset of the LC tasks would be influenced and degrade their quality-of-service (QoS) after a certain HC task exceeds its WCET estimation for LC mode.
3. The third category of strategies, e.g., [48, 49], aim to postpone the occurrence of mode-switch, thereby prolonging the duration for which LC tasks continue to execute.
4. The fourth category of strategies, e.g., [50,51], support the execution of LC tasks via a timely return to LC mode. These studies aim to reduce the duration of time for which the system is in HC mode (and thus reduce the duration of time for which LC jobs are dropped).

3.2.1 Techniques that provide degraded QoS for LC tasks in HC Mode

Strategies belonging to the first category, degrade the QoS of LC tasks in three different forms: C1: reducing the budget allocated to LC tasks, e.g., [21]; C2: extending the periods or deadlines of LC tasks, e.g., [19,20,52,53]; C3: managing the workload of LC tasks at runtime [54,55].

1. The model proposed in [21] allow LC tasks to receive guaranteed but degraded budget when the system stays in HC mode. The authors consider EDF-VD scheduling scheme, and the speedup factor is derived as a function of the ratio between the amount of resource required by different types of tasks in different criticality modes, and reaches 4/3 in the worst case. Besides, the strategy proposed in Chapter 6 of this dissertation also allows LC tasks to continue execution with decreased budget after the system switches to HC mode.
2. In the elastic MC model [52], a minimum level of service is specified for each LC task τ_i in the form of a maximum period, T_i^{MAX} . The system must be MC-schedulable when all HC tasks can

execute up to $C_i(H)$ time units and all LC tasks execute up to $C_i(L)$ time units every T_i^{MAX} time units. At runtime, LC tasks can run more frequently if HC tasks complete before consuming their full budget.

In [20], the elastic MC model is extended to allow LC tasks to have a pair of small and large periods, which represent their service guarantees in LC and HC modes, respectively. The service adaptation strategy on the other hand [19], decreases the dispatch frequency of LC tasks only when an HC task executes beyond $C_i(L)$ time units but does not signal completion. A similar scheme is proposed in [53] for fixed-priority scheduling of dual-criticality systems. At runtime, tasks may adjust their priorities at the mode-switch instant for better system schedulability.

3. Hu et al. [54] proposed a runtime adaptive shaping scheme for MC systems scheduled under the fixed-priority scheduling policy. The incoming workload of LC tasks is dynamically regulated during runtime, using historical knowledge of the workload arrival of HC tasks. Similarly, to protect HC tasks from overload conditions of LC tasks, the work in [55] proposed runtime techniques to monitor and safely switch between feasible activation patterns of LC tasks.

In [17], a complementary scheme is considered to support the execution of LC tasks for fixed-priority scheduling. In this scheme, LC tasks could have their priorities reduced, or have their budget reduced, or have their periods extended after the system switches to HC mode. Thus, this scheme belongs to both C1 and C2.

Note that all the above studies only focus on providing a service (in some cases guaranteed) for all LC tasks after mode-switch. They neither consider any techniques for postponing or avoiding the mode-switch itself (e.g., as proposed in Chapter 6 of this dissertation), nor do they provide any differentiated service to the LC tasks depending on which specific HC task is requesting additional resources (e.g., as proposed in Chapter 5 of this dissertation). Further, all these studies are based on extensions of Vestal's task model, and hence require the designer to specify multiple WCET estimations for each task (as opposed to a single WCET estimation in the dynamic MC model proposed in Chapter 6 of this dissertation).

3.2.2 Techniques to limit the impact of mode-switch

In this section, we present studies that aim to provide differentiated service to the LC tasks depending on which specific HC task is requesting for additional resources. Thus, these techniques are able to limit the impact of mode-switch on LC tasks.

Interference constrained graph (ICG) [46] off-line determines a subset of LC tasks that need to be dropped when a certain HC task violates its WCET estimation for LC mode $C_i(L)$. The budget allocated to these LC tasks will then be used by that HC task to meet its additional execution requirement. A schedulability test which takes into account the interferences between tasks is proposed; this test is based on response time analysis for preemptive fixed-priority scheduling of MC systems [30].

A task grouping strategy which applies a group-based proportionate-fair like scheduling scheme [56] is proposed for the periodic MC task model in [47]. This strategy associates each HC task with a subset of the LC tasks and encapsulates them in a task group, which is scheduled with the other task groups under EDF scheduling policy. When a certain HC task exceeds its WCET estimation for LC mode $C_i(L)$, LC tasks in the other groups are not influenced.

Although the above techniques provide differentiated service to the LC tasks, they do not consider any approaches for postponing or avoiding the mode-switch itself. Further, similar to the studies in the previous section, these techniques also require multiple WCET estimations for each task from the designer. Both these shortcomings are addressed in the dynamic MC task model proposed in Chapter 6 of this dissertation.

Similar to the above two techniques, we propose a component-based MC task model for providing differentiated service to the LC tasks in Chapter 5 of this dissertation. In this work, the component boundaries are used to provide isolation to the LC tasks, thus allowing a subset of them to continue executing even though some HC tasks have already requested for additional resources.

A key advantage of this component-based approach over ICG is the significantly higher schedulability due to the use of EDF-based scheduling as opposed to fixed-priority scheduling in ICG³. When compared to the task grouping strategy, the component-based approach is more flexible because it allows a component to have more than one HC task. This flexibility allows for a more flexible service differentiation for the LC tasks. Finally, unlike our component-based technique, none of the two approaches support a hierarchical scheduling framework, which is essential in the design of open computing environments such as those in automotive (AUTOSAR [9]) and avionics (e.g., ARINC-653 [8]).

3.2.3 Techniques to postpone mode-switches

Another direction to support the execution of LC tasks is to postpone the occurrence of mode-switches, so that the LC tasks can execute for a longer time duration with full service.

³It is not clear how the ICG technique can be extended for EDF-based scheduling, because it is critically dependent on the response time analysis for fixed-priority scheduling.

In [49], a scaling factor $F(> 1)$ is derived so that the system remains MC-schedulable after its budget for LC mode $C_i(L)$ is replaced by $F.C_i(L)$. By scaling the LC budget, the robustness of the system increases and the LC tasks are able to execute up to $C_i(L)$ time units for a longer time duration, because a HC task is less likely to execute beyond $F.C_i(L)$ when compared to $C_i(L)$.

An overrun budgeting (FFOB) mode-switch scheme for MC systems scheduled by EDF is proposed in [48]. This scheme exploits free runtime slacks based on the runtime information of tasks to adaptively postpone the occurrence of mode-switch. Thus, the mode-switch would be triggered only when there is no available slack. To analyze the schedulability of a system, a schedulability test based on the demand bound function (DBF) is proposed.

Both the above techniques aim to postpone or avoid mode-switches by managing the budget of HC tasks. In the first case [49], these budgets are increased based on the available static slack off-line, whereas in the second case [48], these budgets are increased based on the available dynamic slack at runtime. However, these techniques assume that the initial budgets, which are the LC mode WCET estimations for HC tasks, are given by the designer. As discussed in the introduction, this need for multiple WCET estimations increases the burden on the designer, which is not desirable. Further, it also limits the capability of these budget adaptation techniques, because they are now confined by these predefined initial budget values. To overcome these limitations, we propose the dynamic MC task and scheduling model in Chapter 6 of this dissertation. Under this model, the designer is only required to provide a single WCET estimation for each task. Further, task budget allocations are only restricted by an off-line computed total budget for all HC tasks combined as against the individual task budgets in the above studies. This flexibility in budget allocation improves the support for LC tasks as compared to these studies by further postponing mode-switches.

Under the dynamic model, we have developed an efficient runtime budget allocation strategy for HC tasks based on the history of task executions. This is clearly more efficient than the uniform scaling factor F used in [49]. In addition, similar to [48], we have also developed dynamic slack reclamation schemes to further postpone or avoid mode-switches.

3.2.4 Techniques to return to LC Mode

Based on a timely return to LC mode, the negative impact on LC tasks from mode-switches can be reduced. A simple protocol for controlling when the system can switch its execution mode back to LC is to wait until there is an idle instant, and then the system can safely switch back to LC mode. Santy et al. proposed two protocols in [50] that allow for eventually decreasing the execution mode of the system

in order to resume the execution of suspended LC tasks.

A bailout protocol is proposed by Bate et al. [51], where HC tasks take out a loan if they exceed their WCET estimations for LC mode. LC tasks would repay the loan by either executing with a lower service level or not executing at all. After the loan is repaid and a certain condition is satisfied, the system can return to LC mode safely.

These approaches for returning to the LC mode are orthogonal to the techniques developed in this dissertation that either focus on 1) postponing mode-switches, or 2) supporting LC tasks even after mode-switches. Hence, these protocols can be applied in addition to the techniques developed in this dissertation to further provide support for the LC tasks.

Chapter 4

Efficient Schedulability Test for EDF-VD^{*} Scheduling Policy

In this chapter, we present an improved demand bound function (DBF) based schedulability analysis for EDF-VD^{*} scheduling policy. We consider Vestal's multi-criticality model as introduced in Section 2.2 and focus on uniprocessor platforms. In Section 4.1, we present the motivation for this work and the limitations of existing DBF-based analysis for EDF-VD^{*}. Then we present in detail the single-mode (SM) DBF-based test from [25] in Section 4.2. In Sections 4.3 and 4.4, we present our improved Multi-Mode (MM) DBF-based test and virtual deadline tuning algorithm, respectively. Finally, in Section 4.5, we compare the schedulability performance of our proposed technique with the previous works. The techniques presented in this chapter have been previously published in [57]. In Table 4.1, we list the notations used in this chapter.

Table 4.1: Notations Used in Chapter 4

$t^*(m) (m \in \mathbb{N}_+)$	the time instant when the system switches from L(m-1) to L(m) mode.
$C_i(m)$	WCET estimation of τ_i for L(m) mode.
$D_i(m)$	virtual deadline of τ_i for L(m) mode.
$M = \max_{\tau_i \in \tau} \{L_i\}$	total number of criticality levels.
$J_{i,j}$	the j_{th} job of τ_i
$r_{i,j}$	release time of $J_{i,j}$
$d_{i,j}(m) = r_{i,j} + D_i(m)$	absolute deadline of $J_{i,j}$ for L(m) mode.

4.1 Motivation: Limitations of existing DBF-based analyses

Among several scheduling policies and schedulability tests proposed for Vestal's model, the demand bound function (DBF) based analysis for EDF-VD* [25] has exhibited better performance in terms of schedulability than previous works. This work introduced a heuristic virtual deadline tuning algorithm called GreedyTuning, where virtual deadlines can be reduced by different factors. GreedyTuning can increase the chances that a task system is MC-schedulable by EDF-VD*. Using extensive experimental evaluation with synthetic task sets, they show that GreedyTuning outperforms all previous works on mixed-criticality scheduling for a variety of mixed-criticality systems.

GreedyTuning [24] was initially designed for dual-criticality systems and was extended to multi-criticality systems in [25]. However, it suffers from drawbacks in scheduling both dual-criticality and multi-criticality systems. First, it is not good at scheduling task systems with a larger percentage of high-criticality tasks (e.g., see Figure 4.16 in Section 4.5). Besides, it does not scale well with the number of criticality levels, and its schedulability performance drops significantly as the number of criticality levels increases (e.g., see Figure 17 in [25]). The primary reason for these drawbacks is that GreedyTuning analyzes the system in each execution mode independently. It conservatively upper bounds the carry-over workload (i.e., workload from carry-over jobs) when the system behavior switches from a lower criticality level to the one being analyzed.

By considering the system behavior from the start of a busy interval including the mode-switch instant, the analysis in [23] addresses the limitation of GreedyTuning [24], but it is constrained to dual-criticality systems. The first challenge in extending this initial result to more than two criticality levels is that the task execution pattern that can result in worst-case demand for dual-criticality systems is no longer valid in multi-criticality systems. Besides, if we consider the demand from the start of a busy interval as [23] does, there would be multiple mode-switches during the time interval. Then, we have to consider all possible combinations of these mode-switch instants $\{t^*(2), t^*(3), \dots, t^*(M)\}$, and as a result, the complexity of the test is exponential in the number of criticality levels. Also, given a set of these mode-switch instants, the task execution pattern that will result in the worst-case demand depends on all the tasks in the system and their remaining execution time at each of those mode-switches. It is therefore non-trivial to determine a worst-case pattern with low pessimism, and there is no known technique for the same. Thus, there is a need to develop improved and computationally efficient schedulability analysis for multi-criticality MC systems scheduled under EDF-VD*.

Addressing the above drawbacks, the analysis technique presented in this chapter generalizes the

dual-criticality DBF-based test of [23] to multi-criticality systems. It collectively bounds the demand of a MC system τ in $L(m-1)$ and $L(m)$ modes for any arbitrary execution mode m , and as a result does not conservatively bound the carry-over workload as in [25]. It has the same complexity as the dual-criticality test in [23], and thus eliminates the exponential dependence on the number of criticality levels. The virtual deadline assignment algorithm in [25] use a very simple metric to choose candidate tasks and reduce their virtual deadlines. Based on this observation, we also propose a new virtual deadline assignment strategy which uses a combination of several more reasonable parameters as metrics to choose candidate tasks. Finally, through experimental evaluation using synthetic task sets, we demonstrate that the proposed technique significantly outperforms the existing ones for a variety of MC systems.

4.2 SM DBF-Based Test

In this section, we first extend the idea of demand bound function to the MC setting, and introduce the Single-Mode (SM) demand bound function for MC workloads [24]. Test based on this SM DBF has been shown to outperform many studies, e.g., EDF-VD [31] and AMC-max [16], in terms of schedulability.

Let $dbf_{SM}(\tau_i, t, m)$, where $m \in \{1, 2, \dots, M\}$, denote the demand bound function of task τ_i during the time interval $[t^*(m), t^*(m) + t)$. As we can observe, SM DBF only considers the execution demand during $[t^*(m), t^*(m) + t)$, but the system behavior before $t^*(m)$ is ignored. Hence, any schedulability test based on SM DBF must necessarily analyze the system in each execution mode separately. Below we present the SM DBF-based test proposed in [25].

Theorem 3 (Proposition 3 [25]) *A MC task set τ is schedulable by EDF-VD* on a dedicated unit speed uniprocessor platform for all the execution modes if the following conditions hold:*

$$\forall m \in \{1, 2, \dots, M\} : \forall t \in \{1, 2, \dots, t^{max}\} : dbf_{SM}(\tau, t, m) \leq t,$$

where $dbf_{SM}(\tau, t, m) = \sum_{L_i \geq m} dbf_{SM}(\tau_i, t, m)$, and t^{max} is pseudo-polynomial in the size of the input if the utilization of each execution mode is bounded by some constant smaller than 1. Techniques to derive t^{max} are presented in Section 4.1 in [25].

We define condition CN_m^S as follows

$$\forall t \in \{1, 2, \dots, t^{max}\} : \sum_{L_i \geq m} dbf_{SM}(\tau_i, t, m) \leq t$$

Condition CN_m^S captures the schedulability of task set τ in $L(m)$ mode on the assumption that τ is schedulable in $L(m')$ ($m' \in \{1, 2, \dots, m-1\}$) modes.

In $L(1)$ mode, each task τ_i will behave like a normal non-MC task, and all jobs are guaranteed to execute for at most $C_i(1)$ time units. Therefore the dbf for non-MC tasks can be directly applied to capture the demand of τ_i in $L(1)$ mode, i.e.,

$$dbf_{SM}(\tau_i, t, 1) = \left(\left\lfloor \frac{t - D_i(1)}{T_i} \right\rfloor + 1 \right) \times C_i(1). \quad (4.1)$$

If $L_i < m$, then $dbf_{SM}(\tau_i, t, m) = 0$ because tasks with $L_i < m$ will be discarded after $t^*(m)$. On the other hand, if $L_i \geq m$, then we need to consider maximum possible demand of carry-over job $J_{i,k}^*$ during $[t^*(m), d_{i,k}(m)]$. For tasks with $L_i \geq m$, its execution demand comprises two parts: 1) demand of carry-over job $J_{i,k}^*$ and 2) demand of jobs released after $J_{i,k}^*$ but with deadline before $t^*(m) + t$. Execution demand for the carry-over job can then be bounded with the following lemma.

Lemma 1 (Demand of carry-over jobs [24]) *Assume carry-over job $J_{i,k}^*$ with $L_i \geq m$ is scheduled by EDF-VD*, and has remaining scheduling window of x time units until its deadline $d_{i,k}(m) = r_{i,k} + D_i(m)$, as illustrated in Figure 4.1. Then the following hold:*

1. *If $x < D_i(m) - D_i(m-1)$, then $J_{i,k}^*$ must have already finished before $t^*(m)$.*
2. *If $x \geq D_i(m) - D_i(m-1)$, then the execution demand of $J_{i,k}^*$ after $t^*(m)$ is bounded by $C_i(m) - \max\{0, C_i(m-1) - x + D_i(m) - D_i(m-1)\}$.*

4.2.1 Formulating SM DBF

Based on Lemma 1, we formalize SM DBF in this section. The execution demand of τ_i for time interval $[t^*(m), t^*(m) + t]$ is equal to sum of the unfinished execution demand of the carry-over job, and the demand of jobs released after the carry-over job in this interval. The following lemma presents the worst-case job release pattern that maximizes this total demand.

Figure 4.1: Carry-over job $J_{i,k}^*$ has a remaining scheduling window of length x after the system switches to $L(m)$ mode.

Lemma 2 (Maximum demand pattern [25]) *Task τ_i ($L_i \geq m$) can generate maximum execution demand in $L(m)$ mode ($m > 1$) for a time interval length t , when some job has absolute deadline $r_{i,L} + D_i(m)$ at the end of this time interval and all preceding jobs are released as late as possible (as shown in Figure 4.2).*

Under this pattern, the remaining scheduling window of carry-over job $J_{i,k}^*$ is equal to $\text{mod}(t, T_i)$, where $\text{mod}(t, T_i) = t \bmod T_i$. Based on Lemma 1, we can then derive $\text{done}(\tau_i, t, m)$ which denotes the execution demand of the carry-over job that must finish before $t^*(m)$ as follows:

$$\text{done}(\tau_i, t, m) = \begin{cases} \max \{0, C_i(m-1) - \text{mod}(t, T_i) + D_i(m) - D_i(m-1)\} & \text{if } D_i(m) - D_i(m-1) \leq \text{mod}(t, T_i) \text{ and } \text{mod}(t, T_i) < D_i(m) \\ 0 & \text{otherwise} \end{cases} \quad (4.2)$$

Here $\text{mod}(t, T_i)$ is equal to the length of the remaining scheduling window of the carry-over job after $t^*(m)$. Therefore the SM DBF of τ_i in $L(m)$ mode ($m > 1$) can be summarized as follows:

$$\text{dbf}_{SM}(\tau_i, t, m) = \max \left\{ 0, \left(1 + \left\lfloor \frac{t - (D_i(m) - D_i(m-1))}{T_i} \right\rfloor \right) \times C_i(m) \right\} - \text{done}(\tau_i, t, m) \quad (4.3)$$

4.3 Multi-Mode DBF-Based Test For MC Task Systems

From Section 4.2, we can observe that SM DBF considers the behavior of each execution mode separately. That is, it does not use the execution demand in the previous mode (e.g, $L(m-1)$ mode) to determine the remaining execution for carry-over jobs when mode switch happens at $t^*(m)$. If the

 Figure 4.2: Maximum demand of τ_i during $[t^*(m), t^*(m) + t]$

execution load of task set τ in $L(m-1)$ mode is low, then carry-over jobs of many tasks would finish well before their deadlines, and would not generate any demand after $t^*(m)$. This property leads to some interesting results like the fact that SM DBF-based test cannot even schedule some task sets that are schedulable by reservation based approaches (i.e., all tasks are allocated $C_i(L_i)$ units of budget).

For example, task set τ in Table 4.2 is obviously schedulable by reservation based approach since $u_1(2) + u_2(2) < 1$. However, it is not MC-schedulable using SM DBF-based test for any combination of $D_1(1) \in \{1, 2\}$ and $D_2(1) \in \{3, 4, \dots, 15\}$.

Table 4.2: Example task set not schedulable under SM DBF-based test

Task	T_i	$\vec{C}_i$	D_i	L_i
τ_1	2	$\{C_1(1) = 1, C_1(2) = 1\}$	2	2
τ_2	15	$\{C_1(1) = 3, C_2(2) = 7\}$	15	2

The test proposed in [23] considers the system behavior from the start of a busy interval, but is limited to a dual-criticality task system. In such a system, there is at most one mode-switch. However, in a multi-criticality task system, we also have to consider the demand from carry-over jobs released before $t^*(m-1)$ that have deadline after $t^*(m-1)$. As a result, the analysis in [23] is no longer valid for multi-criticality systems.

To extend the DBF-based analysis in [23] to multi-criticality systems, we propose a Multi-Mode (MM) DBF that collectively bounds the demand of τ in $L(m-1)$ and $L(m)$ modes. Suppose the time interval of interest is $[t^*(m-1), t^*(m-1) + t]$, where $t^*(m-1) \leq t^*(m) \leq t^*(m-1) + t$. For simplicity, we denote $t^*(m-1) \equiv 0$, because what determines the total demand is the time interval length $t^*(m) - t^*(m-1)$ and t . That is, the DBF is independent of the exact value of $t^*(m-1)$. Note we still have to consider the demand of carry-over jobs from $L(m-1)$ mode at time instant 0. Thus, the interval becomes $[0, t]$, where $0 \leq t^*(m) \leq t$. Let $dbf_{MM}(\tau, t^*(m), t, m)$ denote the total DBF of task set τ for this time interval.

In fact, if we collectively consider the total demand during more than two execution modes, we can derive an even tighter DBF. For example, we could bound the demand in $L(m)$, $L(m-1)$ and $L(m-2)$ modes collectively (i.e., demand during $[t^*(m-2), t]$, where $t^*(m-2) \leq t^*(m-1) \leq t^*(m) \leq t^*(m-2) + t$). However, in this case, the worst-case demand pattern could be very complex and we need to consider all possible combinations of $t^*(m-1)$, $t^*(m)$ and t . In Section 4.5, we show that the proposed MM DBF-based test already outperforms the SM DBF-based test significantly even though it only considers two successive execution modes.

4.3.1 MM DBF for Jobs

In this section, we first derive MM DBFs for jobs that experience mode-switches. In the time interval of interest $[0, t]$, a job can experience at most two mode-switches (at $t^*(m-1)$ and $t^*(m)$).

Definition 11 We categorize such jobs into four types: $J_{i,X}$ where $X \in \{A, B, C, D\}$. The patterns of these jobs are shown in Figure 4.3.

$J_{i,A}$: $L_i = m-1$, $r_{i,A} < t^*(m-1)$ and $r_{i,A+1} \geq t^*(m-1)$.

$J_{i,B}$: $L_i = m-1$, $r_{i,B} \geq t^*(m-1)$, $r_{i,B} \leq t^*(m)$ and $r_{i,B+1} > t^*(m)$.

$J_{i,C}$: $L_i \geq m$, $r_{i,C} < t^*(m-1)$ and $r_{i,C+1} \geq t^*(m-1)$.

$J_{i,D}$: $L_i \geq m$, $r_{i,D} \geq t^*(m-1)$, $r_{i,D} \leq t^*(m)$ and $r_{i,D+1} > t^*(m)$.

Figure 4.3: Four types of jobs that experience mode-switch.

Let $dbf_{MM}(J_{i,X}, t^*(m), t, m)$, where $X \in \{A, B, C, D\}$, denote an upper bound on the demand of job $J_{i,X}$ during $[0, t]$. Given the value of $r_{i,X}$, we can simply calculate its demand based on the principle that $J_{i,X}$ executes as late as possible as long as it meets its deadline in the previous mode. Therefore, the DBF depends on the virtual deadlines of these jobs, and we use the following four conditions on deadlines to consider various cases.

1) $> t$, it would generate 0 demand. Hence, we have

$$dbf_{MM}(J_{i,B}, t^*(m), t, m) = \begin{cases} \min \{C_i(m-1), t^*(m) - r_{i,B}\} & \text{If (4)} \\ 0 & \text{Otherwise} \end{cases} \quad (4.5)$$

Similar to $J_{i,A}$, if (1) : $d_{i,C}(m-2) < t^*(m-1) \vee \neg(4) : d_{i,C}(m-1) > t$, $J_{i,C}$ generates 0 demand during $[0, t)$. If $\neg(1) \wedge (2)$, $J_{i,C}$ would already finish by $t^*(m)$, and hence would generate demand equal to $C_i(m-1) - C_i(m-2) + \min\{d_{i,C}(m-2) - t^*(m-1), C_i(m-2)\}$. If $\neg(1) \wedge \neg(2) \wedge (3)$, $J_{i,C}$ would generate $C_i(m) - C_i(m-2) + \min\{d_{i,C}(m-2) - t^*(m-1), C_i(m-2)\}$ demand as shown in Figure 4.5. If $\neg(1) \wedge \neg(2) \wedge \neg(3) \wedge (4)$, $J_{i,C}$ would not generate demand after $t^*(m)$ because its deadline $d_{i,C}(m)$ is out of the interval of interest. In this case its demand is bounded by $t^*(m)$. Hence, we have

Figure 4.5: $J_{i,C}$: Case $\neg(1) \wedge \neg(2) \wedge (3)$

$$dbf_{MM}(J_{i,C}, t^*(m), t, m) = \begin{cases} 0 & \text{If (1) } \vee \neg(4) \\ C_i(m-1) - C_i(m-2) + \min \{d_{i,C}(m-2), C_i(m-2)\} & \text{If } \neg(1) \wedge (2) \\ C_i(m) - C_i(m-2) + \min \{d_{i,C}(m-2), C_i(m-2)\} & \text{If } \neg(1) \wedge \neg(2) \wedge (3) \\ \min \{t^*(m), C_i(m-1) - C_i(m-2) + \min \{d_{i,C}(m-2), C_i(m-2)\}\} & \text{If } \neg(1) \wedge \neg(2) \wedge \neg(3) \wedge (4) \end{cases} \quad (4.6)$$

$J_{i,D}$ behaves similar to $J_{i,C}$ except we can ignore condition (1) because it can never be true. Hence, we have

$$dbf_{MM}(J_{i,D}, t^*(m), t, m) = \begin{cases} 0 & \text{If } \neg(4) \\ C_i(m-1) & \text{If } (2) \\ C_i(m) & \text{If } \neg(2) \wedge (3) \\ \min\{C_i(m-1), t^*(m) - r_{i,D}\} & \text{If } \neg(2) \wedge \neg(3) \wedge (4) \end{cases} \quad (4.7)$$

4.3.2 MM DBF for a Single Task ($dbf_{MM}(\tau_i, t^*(m), t, m)$)

In the previous section, we have derived MM DBFs for four types of jobs. Using these DBFs for jobs, we derive MM DBF for a single task. In the time interval $[t^*(m-1) \equiv 0, t)$, tasks with criticality level $L_i = m-1$ could execute during $[0, t^*(m))$, but would be dropped after $t^*(m)$ under EDF-VD* policy. Tasks with criticality level $L_i \geq m$ could execute during the whole time interval $[0, t)$. For L(1) mode (the initial mode), $dbf_{MM}(\tau_i, t^*(0), t, 1)$ can be obtained from the DBF for non-MC tasks (see Equation 4.1).

4.3.2.1 Case When $L_i = m-1$

We first consider tasks with criticality level $L_i = m-1$. According to Definition 11, $J_{i,A}$ is the only job of τ_i released before 0 that can generate demand during $[0, t^*(m))$.

Let $\chi(r_{i,A})$ denote an upper bound of the demand that τ_i generates during $[0, t^*(m))$, given the value of $r_{i,A}$ and assuming all jobs are released as soon as possible with period T_i . Given $r_{i,A}$, there will be at most

$$n_{m-1} = \left\lfloor \frac{t^*(m) - (r_{i,A} + T_i)}{T_i} \right\rfloor$$

jobs released during $[0, t^*(m))$. Here n_{m-1} denotes the number of jobs released between $J_{i,A}$ and $J_{i,B}$. If $r(J_i^A) + T_i > t^*(m)$, according to the definition of $J_{i,A}$ in Definition 11, $J_{i,A}$ is the only job from τ_i that generates demand during $[0, t)$. In this case, $\chi(r_{i,A})$ is equal to $dbf_{MM}(J_{i,A}, t^*(m), t, m)$. In sum, we have

$$\chi(r_{i,A}) = \begin{cases} dbf_{MM}(J_{i,A}, t^*(m), t, m) & \text{if } r_{i,A} + T_i > t^*(m) \\ dbf_{MM}(J_{i,A}, t^*(m), t, m) + n_{m-1}C_i(m-1) + dbf_{MM}(J_{i,B}, t^*(m), t, m) & \text{otherwise} \end{cases} \quad (4.8)$$

If all tasks have integer release times, we get

$$dbf_{MM}(\tau_i, t^*(m), t, m) = \max_{r_{i,A} \in \{0, -1, -2, \dots, -T_i\}} \{\chi(r_{i,A})\}$$

For more generic cases, the following lemma defines the release pattern when τ_i ($L_i = m - 1$) generates maximum demand during $[0, t^*(m))$.

Figure 4.6: $r_{i,A} = -D_i(m - 2) + C_i(m - 2)$

Figure 4.7: $r_{i,A} = \text{mod}(t - D_i(m - 1), T_i) - T_i$

Lemma 3 Task τ_i with $L_i = m - 1$ generates maximum demand during $[0, t^*(m))$ if $r_{i,A} = -D_i(m - 2) + C_i(m - 2)$ or $r_{i,A} = \text{mod}(t - D_i(m - 1), T_i) - T_i$ (i.e., deadline $d_{i,A}(m - 1) = t \vee d_{i,B}(m - 1) = t$), and all the jobs are released as soon as possible with period T_i . These two patterns are shown in Figures 4.6 and 4.7, respectively.

Proof: **(C1):** If $t \leq D_i(m - 1)$, obviously the demand of τ_i is maximized when $r_{i,A} = \text{mod}(t - D_i(m - 1), T_i) - T_i$, i.e., $d_{i,A}(m - 1) = t$ as shown in Figure 4.8.

(C2): If $(t^*(m) \leq C_i(m - 1) \wedge t > D_i(m - 1))$, we can find the demand of τ_i is maximized when $r_{i,A} = -D_i(m - 2) + C_i(m - 2)$, which is bounded by $t^*(m)$ as shown in Figure 4.9. Now we can exclude the above two cases, and we only need to consider the case when $t > D_i(m - 1) \wedge t^*(m) > C_i(m - 1)$.

(C3): $r_{i,B} + D_i(m - 1) \leq t$ when $r_{i,A} = -D_i(m - 2) + C_i(m - 2)$ as shown in Figure 4.10. As

 Figure 4.8: Case when $t \leq D_i(m-1)$

 Figure 4.9: Case when $t^*(m) \leq C_i(m-1) \wedge t > D_i(m-1)$

Figure 4.10: Case C3

we shift the release pattern left, i.e., $r_{i,A} = -D_i(m-2) + C_i(m-2) + x | x < 0$, the demand change of $J_{i,A}$, $\Delta_1(x)$, is of the following form.

$$\Delta_1(x) = \begin{cases} x & \text{If } x \in [-C_i(m-2), 0) \\ -C_i(m-1) & \text{If } x \in [-C_i(m-2) + D_i(m-2) - T_i, -C_i(m-2)) \end{cases}$$

Meanwhile the demand of $J_{i,B}$ will at most increase linearly, and hence the total demand of τ_i will stay the same or decrease as we shift the pattern left. As we shift the release pattern right, the demand

4.3.2.2 Case When $L_i \geq m$

In this section, we consider tasks with criticality level $L_i \geq m$. According to the definition of $J_{i,C}$ in Definition 11, $J_{i,C}$ is the only job released before 0 that can generate demand during $[0, t)$.

Let $\psi(r_{i,C})$ denote an upper bound on the demand that τ_i generates during $[0, t)$, given the value of $r_{i,C}$ and assuming all jobs are released as soon as possible with period T_i . Let n_{m-1} denote the number of jobs released during $[t^*(m-1), r_{i,D})$ (i.e., jobs released between $J_{i,C}$ and $J_{i,D}$), and n_m denote the number of jobs with release time $r_i > t^*(m)$ (i.e., released after $J_{i,D}$) and deadline $d_i(m) \leq t$. Given the value of $r_{i,C}$,

$$n_{m-1} = \left\lfloor \frac{t^*(m) - r_{i,C} - T_i}{T_i} \right\rfloor.$$

The first job released after $J_{i,D}$, i.e., $J_{i,D+1}$, is released at $r_{i,C} + (n_{m-1} + 2) \times T_i$, and hence

$$n_m = \left\lfloor \frac{t - r_{i,D+1} - D_i(m)}{T_i} \right\rfloor + 1.$$

Note that if $n_{m-1} < 0$, i.e., $t^*(m) < r_{i,C} + T_i$, it implies there does not exist $J_{i,D}$. Therefore we have

$$\psi(r_{i,C}) = \begin{cases} dbf_{MM}(J_{i,C}, t^*(m), t, m) + n_{m-1}C_i(m-1) + dbf_{MM}(J_{i,D}, t^*(m), t, m) + n_m C_i(m) & \text{if } t^*(m) \geq r_{i,C} + T_i \\ dbf_{MM}(J_{i,C}, t^*(m), t, m) + n_m C_i(m) & \text{if } t^*(m) < r_{i,C} + T_i \end{cases} \quad (4.10)$$

Figure 4.12: Job release pattern for tasks with $L_i \geq m$

If all tasks have integer release times, we get

$$dbf_{MM}(\tau_i, t^*(m), t, m) = \max_{r_{i,C} \in \{0, -1, -2, \dots, -T_i\}} \{\psi(r_{i,C})\}$$

For more generic cases, the following lemma defines the release pattern so that τ_i with $L_i \geq m$ generates maximum demand during $[0, t)$.

Lemma 4 *Task τ_i with $L_i \geq m$ generates maximum demand during $[0, t)$ if $r_{i,C} = \text{mod}(t - D_i(m), T_i) - T_i$ (i.e., the last job released before t has $d_{i,L}(m) = t$), $r_{i,C} = \text{mod}(t - D_i(m - 1), T_i) - T_i$ (i.e., $d_{i,L}(m - 1) = t$) or $r_{i,C} = -D_i(m - 2) + C_i(m - 2)$, and all the jobs are released as soon as possible with period T_i .*

Proof: **(C1):** If $t - t^*(m) > D_i(m)$, the demand of $\tau_i | L_i \geq m$ during $[0, t)$ is maximized when $r_{i,C} = \text{mod}(t - D_i(m), T_i) - T_i$ (the last job released before t , J_i^L , has $d_{i,L}(m) = t$) as shown in Figure 4.13. This is because as we shift the release pattern right, the demand of J_i^L would decrease from

Figure 4.13: Case when $t - t^*(m) > D_i(m)$

$C_i(m)$ to 0 while the increase in demand of other jobs including $J_{i,C}$ and $J_{i,D}$ is bounded by $C_i(m)$. On the other hand, as we shift the release pattern left, the total demand of all jobs would decrease or stay the same.

(C2): If $t - t^*(m) < D_i(m) - D_i(m - 1)$, no job of τ_i could execute for more than $C_i(m - 1)$ time units (no job will execute after $t^*(m)$). Thus τ_i behaves as a task with $L_i = m - 1$, and from Lemma 3 we know the demand of τ_i is maximized when $r_{i,C} \in \{-D_i(m - 2) + C_i(m - 2), \text{mod}(t - D_i(m - 1), T_i) - T_i\}$.

(C3): If $D_i(m) - D_i(m - 1) \leq t - t^*(m) \leq D_i(m)$, at most one job (either $J_{i,C}$ or $J_{i,D}$) could generate execution demand as much as $C_i(m)$ as shown in Figure 4.14. Thus suppose initially $r_{i,C} = \text{mod}(t - D_i(m), T_i) - T_i$, i.e., the scenario of Figure 4.14. If we shift the release pattern left, the demand of τ_i would only decrease or stay the same. On the other hand, as we shift the release pattern right, no job would execute after $t^*(m)$, and hence τ_i behaves as a task with $L_i = m - 1$. Therefore from Lemma 3 we know the demand of τ_i is maximized when $r_{i,C} \in \{-D_i(m - 2) + C_i(m - 2), \text{mod}(t - D_i(m - 1), T_i) - T_i\}$.

 Figure 4.14: Case when $D_i(m) - D_i(m-1) \leq t - t^*(m) \leq D_i(m)$

In summary, the demand of $\tau_i | L_i \geq m$ during $[0, t)$ is maximized if $r_{i,C} = -D_i(m-2) + C_i(m-2)$ or $r_{i,C} = \text{mod}(t - D_i(m-1), T_i) - T_i$ or $r_{i,C} = \text{mod}(t - D_i(m), T_i) - T_i$. ■

Using Lemma 4, we define $dbf_{MM}(\tau_i, t^*(m), t, m)$ for task τ_i with $L_i \geq m$ as follows.

$$dbf_{MM}(\tau_i, t^*(m), t, m) = \max_{r_{i,C} \in \{\text{mod}(t - D_i(m), T_i) - T_i, -D_i(m-2) + C_i(m-2), \text{mod}(t - D_i(m-1), T_i) - T_i\}} \{\psi(r_{i,C})\} \quad (4.11)$$

4.3.3 MM DBF-based Schedulability Test

Once we derive the MM dbf for each task, we can simply add up the demand of all tasks in the system to get the dbf for task set τ .

$$dbf_{MM}(\tau, t^*(m), t, m) = \sum_{L_i \geq m-1} dbf_{MM}(\tau_i, t^*(m), t, m) \quad (4.12)$$

In fact, we can also apply a simple optimization technique to further tighten the total demand. Note that the demand of τ_i ($L_i \geq m$) during $[t^*(m), t)$ would not exceed $dbf_{SM}(\tau_i, t - t^*(m), m)$, because $dbf_{SM}(\tau_i, t - t^*(m), m)$ upper bounds the demand of τ_i for a time interval length $t - t^*(m)$ in L(m) mode. Besides, the total execution demand of τ during $[0, t^*(m))$ cannot exceed $t^*(m)$. Thus, based on these two facts, we have the following theorem.

Theorem 4

$$\begin{aligned}
 dbf_{MM}(\tau, t^*(m), t, m) = & \underbrace{\sum_{L_i \geq m} dbf_{SM}(\tau_i, t - t^*(m), m)}_{\text{this expression is denoted as EXP H}} \\
 + \min & \left\{ t^*(m), \underbrace{\sum_{L_i \geq m} (dbf_{MM}(\tau_i, t^*(m), t, m) - dbf_{SM}(\tau_i, t - t^*(m), m)) + \sum_{L_i = m-1} dbf_{MM}(\tau_i, t^*(m), t, m)}_{\text{this expression is denoted as EXP L}} \right\}
 \end{aligned} \tag{4.13}$$

Proof: If $\text{EXP L} \leq t^*(m)$, then the optimization does not work at all. Otherwise, if $\text{EXP L} > t^*(m)$, then EXP H and $t^*(m)$ upper bound the total demand during $[t^*(m), t]$ and $[0, t^*(m))$, respectively. Thus, Equation 4.13 still upper bounds the total demand of τ during $[0, t]$. ■

Once we construct $dbf_{MM}(\tau, t^*(m), t, m)$ for $m \in \{1, 2, \dots, M\}$, we can compare the execution demand of τ with the amount of available execution time (i.e., t) to check the schedulability of τ . The following theorem is a straightforward extension of Theorem 3 in Section 4.2, and can be used to determine whether a task set τ is schedulable by EDF-VD* on a dedicated unit speed uniprocessor platform.

Theorem 5 *A MC task set τ is schedulable by EDF-VD* on a dedicated unit speed uniprocessor platform for all the execution modes if the following conditions hold:*

$$\forall m \in \{1, 2, \dots, M\} : \forall t \in \{1, 2, \dots, t^{max}\} : \forall t^*(m) \in \{0, 1, 2, \dots, t\} : dbf_{MM}(\tau, t^*(m), t, m) \leq t \tag{4.14}$$

where t^{max} is pseudo-polynomial in the size of the input if the system utilization of each execution mode is bounded by some constant smaller than 1.

Proof: Suppose the above conditions in Equation 4.14 hold but τ still misses deadline at the end of a busy time interval $[t_S, t_F]$ and the system is in $L(m)$ mode at t_F . Since there is a deadline miss at t_F , the execution demand of τ during $[t_S, t_F]$ must be greater than $t_F - t_S$. Since the demand during $[t_S, t^*(m-1)]$ cannot exceed $t^*(m-1) - t_S$, the execution demand of τ during $[t^*(m-1), t_F]$ must also be greater than $t_F - t^*(m-1)$. However, this contradicts the condition $dbf_{MM}(\tau, t^*(m), t_F - t^*(m-1), m) \leq t_F - t^*(m-1)$. An upper bound for t^{max} is derived in Section 4.3.3.1. ■

4.3.3.1 Upper Bound for t^{max}

In this section, we derive an upper bound for t^{max} . For task τ_i with $L_i = m - 1$, we can observe that its demand during $[0, t^*(m))$ is upper bounded by $(\frac{t^*(m) - 2 \times C_i(m-1)}{T_i} + 2) \times C_i(m-1)$. For task τ_i with $L_i \geq m$, its demand during $[0, t)$ is upper bounded by

$$\begin{aligned}
 & \underbrace{C_i(m-1)}_{\geq dbf_{MM}(J_{i,C}, t^*(m), t, m)} + \underbrace{\frac{t^*(m) - C_i(m-1)}{T_i} \times C_i(m-1)}_{\geq n_{m-1} \times C_i(m-1)} + \underbrace{C_i(m)}_{\geq dbf_{MM}(J_{i,D}, t^*(m), t, m)} \\
 & + \underbrace{\frac{t - t^*(m) - D_i(m) + T_i}{T_i} \times C_i(m)}_{\geq n_m \times C_i(m)} \\
 & = C_i(m-1) \times \frac{t^*(m) - C_i(m-1) + T_i}{T_i} + \frac{t - t^*(m) - D_i(m) + 2 \times T_i}{T_i} \times C_i(m)
 \end{aligned}$$

Suppose $dbf_{MM}(\tau, t^*(m), t, m) > t$, then it must be that

$$\begin{aligned}
 & \sum_{L_i \geq m} \left(C_i(m-1) \times \frac{t^*(m) - C_i(m-1) + T_i}{T_i} + \frac{t - t^*(m) - D_i(m) + 2 \times T_i}{T_i} \times C_i(m) \right) \\
 & + \sum_{L_i = m-1} \left(\frac{t^*(m) + 2 \times (T_i - C_i(m-1))}{T_i} \right) \times C_i(m-1) \\
 & = t^*(m) \times \underbrace{\left(\sum_{L_i = m-1} \frac{C_i(m-1)}{T_i} + \sum_{L_i \geq m} \frac{C_i(m-1) - C_i(m)}{T_i} \right)}_{\text{Exp A}} + \underbrace{\sum_{L_i = m-1} \frac{T_i - C_i(m-1)}{T_i} \times 2 \times C_i(m-1)}_{\text{Exp B}} \\
 & + \underbrace{\left(\sum_{L_i \geq m} \frac{2 \times T_i - D_i(m)}{T_i} \times C_i(m) + \frac{T_i - C_i(m-1)}{T_i} \times C_i(m-1) \right)}_{\text{Exp C}} + \sum_{L_i \geq m} \frac{C_i(m)}{T_i} \times t > t
 \end{aligned}$$

If $\text{Exp A} > 0$, the total demand is maximized if $t^*(m) = t$, and else if $\text{Exp A} \leq 0$, the total demand is maximized if $t^*(m) = 0$. Therefore the value of t is bounded by

$$(\text{Exp B} + \text{Exp C}) / \underbrace{\left(1 - \sum_{L_i \geq m} \frac{C_i(m)}{T_i} \right)}_{>0}$$

or

$$(\text{Exp B} + \text{Exp C}) / \underbrace{\left(1 - \sum_{L_i \geq m-1} \frac{C_i(m-1)}{T_i} \right)}_{>0}$$

4.4 Virtual Deadline Assignment Algorithm

In the previous section, we have introduced the MM DBF-based test in which virtual deadline for each mode plays a key role in shaping the demand of carry-over jobs. The choice of virtual deadline for each task therefore has a significant impact on the performance of the proposed schedulability test. We can decrease τ_i 's demand in $L(m)$ ($m > 1$) mode at the cost of increasing the demand in previous modes. By choosing suitable values for $D_i(m-1)$ for each τ_i , we can increase the chances that τ is schedulable in $L(m)$ mode.

The process of finding suitable values for the virtual deadlines is very challenging, because it is infeasible to try all possible combinations of $D_i(m)$ for all the tasks in different execution modes. In this section, we propose a heuristic virtual deadline tuning algorithm which has pseudo-polynomial time complexity, as long as the system utilization of each execution mode is bounded by some constant smaller than 1. This algorithm comprises three parts:

1. Algorithm 1: $\text{TuneMode}(m)$ checks whether a task set is schedulable in $L(m)$ mode, under the assumption that τ is schedulable in all previous modes.
2. Algorithm 2: $\text{TuneSystem}(\tau)$ applies $\text{TuneMode}(m)$ on all the execution modes starting from $L(M)$ mode and proceeding in a reverse order, until it has successfully tuned the deadlines in all execution modes or failed to do so in some execution mode.
3. Algorithm 3: $\text{find_candidate_m}(\Psi, t)$ is used by $\text{TuneMode}(m)$ to select a candidate task for deadline tuning.

4.4.1 Algorithm 1: $\text{TuneMode}(m)$

In Sections 4.2 and 4.3, we introduced the existing SM DBF-based schedulability test and our proposed MM DBF-based test, respectively. An MC task system τ is schedulable in $L(m)$ mode assuming no deadline miss happens in previous execution modes if one of the following two condition holds.

1. CN_m^S : $\forall t \in \{1, 2, \dots, t^{max}\} : dbf_{SM}(\tau, t, m) \leq t$ (Theorem 3).
2. CN_m^M : $\forall t \in \{1, 2, \dots, t^{max}\} : \forall t^*(m) \in \{1, 2, \dots, t\} : dbf_{MM}(\tau, t^*(m), t, m) \leq t$ (Theorem 5).

Although the complexity of both the above conditions is pseudo-polynomial, there is a quadratic increase in complexity to check condition CN_m^M . For a certain t^{max} , while the SM DBF-based test [25] needs to calculate the demand for t^{max} times, the MM DBF-based test needs to calculate the demand for $t^{max} \times t^{max}$ times.

Therefore, as shown in lines 18-26 of Algorithm 1, we use the condition CN_m^M only as a complement to condition CN_m^S . Suppose condition CN_m^S fails at a certain t . Then we will check whether condition CN_m^M is satisfied for that time interval length. Thus, once we find a t_F such that $dbf_{SM}(\tau, t_F, m) > t_F$ (line 16 of Algorithm 1), we use the MM DBF to check whether the following holds.

$$\forall t \in \{t_F, t_F + 1, \dots, t^{max}\} : dbf_{MM}(\tau, t^*(m) = t - t_F, t, m) \leq t$$

If the above inequalities hold, then it means no deadline miss happens t_F time units after $t^*(m)$ and we can go to the next iteration (lines 24-26 of Algorithm 1).

Otherwise if both CN_m^M and CN_m^S fail, a candidate task will be chosen according to Algorithm 3 and its virtual deadline $D_i(m-1)$ is reduced by 1 (lines 30-31 of Algorithm 1). By doing this, we can reduce the demand of these tasks in L(m) mode. We do this without considering the schedulability in L(m-1) mode itself, hoping that τ in L(m-1) mode can later be made schedulable by decreasing deadlines for L(m-2) mode, i.e., $D_i(m-2)$. After the deadline of a candidate task is successfully tuned by one unit, the flag variable “changed” is set to True and we will jump out of the FOR loop (lines 35-38). Once $D_i(m-1)$ decreases to $C_i(m-1)$, τ_i is eliminated from the candidate set Ψ_m (lines 32-35 of Algorithm 1), and then we need to find another candidate task. We repeat the above steps until TuneMode(m) returns true or the candidate task set Ψ_m becomes empty.

An exceptional scenario is that if we decrease $D_i(1)$ to reduce the demand in L(2) mode, τ may become unschedulable in L(1) mode. However there does not exist $D_i(0)$, and hence we cannot tune $D_i(0)$ to reduce the demand in L(1) mode. As shown from lines 6-15 of Algorithm 1, once τ is not MC-Schedulable after we tune $D_i(1)$, we will undo the changes to deadlines, i.e., $D_i(1) = D_i(1) + 1$, and then the candidate task τ_i is removed from Ψ_m .

4.4.2 Algorithm 2: TuneSystem(τ)

Algorithm 2 applies Algorithm 1, i.e., TuneMode(m), on all the execution modes starting from L(M) mode and proceeding in a reverse order, until it has successfully tuned the deadlines in all execution modes or failed to do so in some criticality mode. Therefore, the complexity of TuneSystem(τ) increases

Algorithm 1 TuneMode(m)

```

1:  $\Psi_m \leftarrow \{\tau_i | L_i \geq m\}$ 
2:  $changed \leftarrow \text{False}$ 
3: while  $changed = \text{True}$  do
4:    $changed \leftarrow \text{False}$ 
5:   for  $t_1 \in \{1, 2, \dots, t^{max}\}$  do
6:     if  $m = 2$  and  $dbf_{SM}(\tau, t_1, 1) > t_1$  then
7:       if  $changed = \text{True}$  then
8:          $D_i(1) \leftarrow D_i(1) + 1$ 
9:          $\Psi_m.remove(\tau_i)$ 
10:         $changed \leftarrow \text{True}$ 
11:        break
12:      end if
13:    else
14:      return False
15:    end if
16:    if  $dbf_{SM}(\tau, t_1, m) > t_1$  then
17:       $CN_m^M \leftarrow \text{True}$ 
18:      for  $t_2 \in \{t_1, t_1 + 1, \dots, t^{max}\}$  do
19:        if  $dbf_{MM}(\tau, t_2 - t_1, t_2) > t_2$  then
20:           $CN_m^M \leftarrow \text{False}$ 
21:          break
22:        end if
23:      end for
24:      if  $CN_m^M = \text{True}$  then
25:        continue
26:      end if
27:      if  $\Psi_m = \emptyset$  then
28:        return False
29:      end if
30:       $\tau_i \leftarrow \text{find\_candidate\_m}(\Psi_m, t)$  (Algorithm 3)
31:       $D_i(m - 1) \leftarrow D_i(m - 1) - 1$ 
32:      if  $D_i(m - 1) < C_i(m - 1)$  then
33:         $D_i(m - 1) \leftarrow D_i(m - 1) + 1$ 
34:         $\Psi_m.remove(\tau_i)$ 
35:      else
36:         $changed \leftarrow \text{True}$ 
37:        break
38:      end if
39:    end if
40:  end for
41: end while return True

```

linearly as the number of levels increase, because the complexity of TuneMode(m) is independent of the value of m . However, since TuneMode($m=2$) also checks the schedulability of τ in L(1) mode, in

TuneSystem(τ), we do not need to consider TuneMode($m=1$).

Algorithm 2 TuneSystem(τ)

```

1:  $M \leftarrow \max_{\tau_i \in \tau} \{L_i\}$ 
2: for  $m \in \{M \text{ to } 2\}$  do
3:   if TuneMode( $m$ ) = False then
4:     return False
5:   end if
6: end for
7: return True
    
```

4.4.3 Algorithm 3: find_candidate_m(Ψ, t)

To select a candidate task for deadline tuning in each iteration, TuneMode(m) uses Algorithm 3 (find_candidate_m(Ψ, t)). Note that, GreedyTuning [24] uses a very simple metric to choose a candidate task, i.e., τ_i with the maximum

$$\Delta_i = dbf_{SM}(\tau_i, t, m) - dbf_{SM}(\tau_i, t - 1, m)$$

is always chosen as a candidate task.

To maximize schedulability in L(m) mode by reducing the demand of a candidate task, Δ_i is the most intuitive metric. However, there are many other parameters which are also important in choosing a good candidate task. Therefore, in Algorithm 3, we extend this metric to additionally consider other factors in candidate selection such as the impact of change in virtual deadline on the schedulability for previous mode, i.e., L($m-1$) mode. In Section 4.5, we show that the new metrics for choosing a candidate task outperform the one in [25].

When a candidate task τ_i 's virtual deadline $D_i(m-1)$ decreases by one, the impact on the demand of L($m-1$) mode is different. Suppose task set τ has tasks τ_1 and τ_2 , where $T_1 = D_1(2) = D_1(1) = 10000$ and τ_2 has $T_2 = D_2(2) = D_2(1) = 2$. Also, suppose L(2) mode of τ is currently not schedulable but can be tuned to become schedulable if either $D_1(1)$ or $D_2(1)$ decreases by one. The impact of $D_1(1) \leftarrow D_1(1) - 1$ or $D_2(1) \leftarrow D_2(1) - 1$ is different, i.e.,

$$\frac{\Delta D_1(1)}{D_1(1)} = 0.0001 \ll \frac{\Delta D_2(1)}{D_2(1)} = 0.5$$

.

If $D_1(1) \leftarrow D_1(1) - 1$, the system demand of L(1) mode stays the same for any time interval

length $t (< 9999)$. As a result, the $L(1)$ mode is easier to be MC-schedulable compared to the case if $D_2(1)$ decreases to 1. From the above discussion, we know a task with larger Δ_i and D_i^{m-1} is more likely to become a better candidate task. Therefore, among all possible candidate tasks in Ψ_m , we use $\Delta_i \times D_i^{m-1}$ as the main metric to choose a candidate task.

Until now, we did not consider the case when all tasks have $\Delta_i = 0 \Rightarrow \Delta_i \times D_i(m-1) = 0$. This means that there does not exist a candidate task so that the total demand would decrease if $D_i(m-1) \leftarrow D_i(m-1) - 1$. As shown in Figure 4.2, if $\Delta_i = 0$, it must be that

$$LEN_i = \text{mod}(t, T_i) - (D_i(m) - D_i(m-1)) - C_i(m-1) > 0.$$

The demand of τ_i would start to decrease if $D_i(m-1) \leftarrow D_i(m-1) - LEN_i - 1$. Therefore, we also consider $\min\{\max\{0, LEN_i\}\}$ among the candidate tasks for selection. Hence, in Algorithm 3, we sort tasks in Ψ_m with first key $\Delta_i \times D_i(m-1)$ in ascending order and then with second key $\max\{0, LEN_i\}$ in descending order.

Algorithm 3 find_candidate_m(Ψ, t)

```

1:  $Q \leftarrow$  empty queue
2: for  $\tau_i \in \Psi$  do
3:    $Q.\text{insert}(\Delta_i \times D_i(m-1), \max\{0, LEN_i\}, \tau_i)$ 
4: end for
5:  $Q.\text{sort}(\text{ascending}, \text{descending})$   $\triangleright$  sort tasks in  $\Psi$  with first key  $\Delta_i \times D_i(m-1)$  in ascending order
   and then with second key  $\max\{0, LEN_i\}$  in descending order
6:  $\tau_i \leftarrow Q.\text{pop}()$   $\triangleright$  return the corresponding task
7: return  $\tau_i$ 
    
```

4.5 Evaluation

In this section we evaluate the ability of the proposed strategy in terms of acceptance ratios, i.e., the fraction of schedulable task sets. We aim to compare with the existing work named GreedyTuning [25] based on SM DBF. From extensive experiments, it has already been shown in [25] that this work outperforms existing studies (e.g., [14, 31, 32]) for a variety of generic MC systems. Therefore, through comparison with this SM DBF work, we aim to show that the proposed techniques also outperform those studies, including the one based on SM DBF-based test.

We will study the impact of varying parameters of tasks on the acceptance ratios of the following approaches: 1): GreedyTuning (GT) [25], 2): SM DBF-based test from [25] with the improved deadline

assignment strategy (i.e, Algorithm 3) (GTI), 3): our improved MM DBF-based test (IMPT).

4.5.1 Task Set Generation

Suppose task set τ is a empty task set ($\tau = \emptyset$) initially. Randomly generated tasks are added to the task set τ repeatedly until certain requirements are met. The parameters of each task is controlled by the following parameters.

- $P(m)$ denotes the probability that τ_i has $L_i = m$.
- $C_i(1)$ is drawn using an uniform distribution over $[1, 10]$.
- RC_m denotes the maximum ratio of $C_i(m)/C_i(m-1)$.
- $C_i(m)$ is drawn using an uniform distribution over $[C_i(m-1), RC_m \times C_i(m-1)]$.
- T_i is drawn using an uniform distribution over $[C_i(L_i), 200]$.
- D_i is drawn using an uniform distribution over $[D_i^{MIN}, T_i]$, where $D_i^{MIN} = \lfloor C_i(L_i) + RD \times (T_i - C_i(L_i)) \rfloor$. Here $RD \in [0, 1]$ is a parameter that determines the minimal value of deadline D_i .

Let $U_\tau = \max_{m \in \{1, 2, \dots, M\}} \{U(m)\}$ denote the utilization bound of a task set τ . For a given utilization bound $Ubound$, our generation procedure requires U_τ to fall within the small interval between $[Ubound - \epsilon, Ubound]$ ($\epsilon = 0.005$). As long as $U_\tau < Ubound - \epsilon$, a new task will be randomly generated and added to τ . Once U_τ falls within the range $[Ubound - \epsilon, Ubound]$, the generation procedure for τ is considered complete. However if U_τ becomes greater than $Ubound$ after a new task τ_i is added to τ , we discard the whole task set and start with a new empty task set.

4.5.2 Evaluation of Dual-Criticality Systems

In this section, we present evaluation results for the generated dual-criticality task sets. Figure 4.15 shows the acceptance ratio as a function of utilization ($Ubound$) of task sets generated with $RC_2 = 3$, $P(1) = P(2) = 0.5$ and $RD \in \{0.5, 1\}$. Each data is based on 1000 generated task sets. As shown in Figure 4.15, our improved MM DBF based schedulability test (IMPT) strictly outperforms GreedyTuning (GT). However the performance gap between GT and IMPT is very small when $RD = 1$, which implies GT already does quite well in scheduling dual-criticality implicit deadline task systems.

 Figure 4.15: Dual-Criticality Task Systems with $RD \in \{0.5, 1\}$ and $P(2) = 0.5$

Figure 4.16 shows the acceptance ratios of task sets generated with $RD = 0.5$, $RC_2 = 3$ and $P(2) \in \{0.25, 0.75\}$. The acceptance ratios of task sets with $RD = 0.5$, $RC_2 = 3$ and $P(2) = 0.5$ can be found in Figure 4.15. We can observe that as $P(2)$ increases from 0.25 to 0.75, the acceptance ratio of GT drops quickly. Even though the SM DBF-based test with the improved deadline assignment strategy (GTI) always has a higher acceptance ratio than GT, its acceptance ratio is much lower than IMPT when $P(2) = 0.75$. One interpretation for this trend is that SM DBF-based test is not good at scheduling systems with a larger percentage of high-criticality tasks. On the other hand, the acceptance ratio of IMPT drops much slower, and its acceptance ratio becomes almost two times as much as GT when $P(2) = 0.75$ and $U_{bound} = 0.95$. In fact, the acceptance ratio of IMPT when $P(2) = 0.75$ is closer to its acceptance ratio when $P(2) = 0.5$.

4.5.3 Evaluation of Multi-Criticality Systems

In this section we compare the acceptance ratios of different tests for multi-criticality task systems. Figure 4.17 shows the acceptance ratios of systems with three criticality levels, $P(1) = P(2) = P(3) = 1/3$, $RC_2 = RC_3 = 2$ and $RD \in \{0.5, 1\}$. As we can observe, the performance of all the three

 Figure 4.16: Dual-Criticality Task Systems with $RD = 0.5$ and $P(2) \in \{0.25, 0.75\}$

approaches drops, but IMPT outperforms GT even for small utilization bounds.

To study how the acceptance ratio changes as the number of criticality levels increases, we also present the weighted acceptance ratios of different approaches for each criticality level. Suppose $A(U)$ is the acceptance ratio of a certain approach for utilization $U \in [0.55, 0.6, 0.65, \dots, 0.95]$. Then, the weighted acceptance ratio is defined as

$$\frac{\sum_{U \in \{0.55, 0.6, 0.65, \dots, 0.95\}} A(U) \times U}{\sum_{U \in \{0.55, 0.6, 0.65, \dots, 0.95\}} U}.$$

In Figure 4.18, we plot the weighted acceptance ratio as a function of number criticality levels in each task set from 2 to 5. As we can observe in Figure 4.18, the gap in weighted acceptance ratio between GT and IMPT is in fact quite small when the number of criticality level in a task set is equal to 2. However, this gap becomes much larger when the number of criticality level increases. The acceptance ratio of IMPT also becomes lower (especially for the case when $RD = 0.5$) as the number of criticality level increases. One interpretation for this trend is that there is not enough space to tune the virtual deadlines since we need to set multiple virtual deadlines for each task. However, note that, it does not mean that the proposed approach IMPT has very poor performance, and can schedule only a small portion of

Figure 4.17: Task Systems with Three Criticality Levels

generated task sets. Since there is no known exact feasibility test for MC task systems, we are unable to eliminate all the infeasible task sets from our experiments. Nevertheless, we can still conclude that the proposed approach outperforms GT.

From the evaluation results in this section, we can conclude that IMPT strictly outperforms GT in terms of acceptance ratios. In fact, the gap of weighted acceptance ratio between IMPT and GT is quite stable as the number of criticality levels increases. However, IMPT still suffers from performance drop, and this problem gets worse for constrained deadline task sets.

4.6 Conclusions and Future Work

The SM DBF-based test for EDF-VD* [25] has exhibited better performance in terms of schedulability than previous works, but still has its own limitations in scheduling both dual-criticality and multi-criticality systems. On the other hand, the analysis in [23] addresses its limitation in scheduling dual-criticality systems, but it cannot be directly extended for multi-criticality systems.

In this chapter, we addressed the above limitations and proposed the MM DBF-based schedulability

 (a) $RD = 1$

 (b) $RD = 0.5$

Figure 4.18: Weighted Acceptance Ratios

test and corresponding virtual deadline assignment strategy for multi-criticality systems scheduled under EDF-VD*. This test can avoid the problems of SM DBF-based test by collectively considering the demand generated in successive execution modes. However, this performance improvement comes at the cost of a quadratic increase in the test complexity. Note that the complexity is still independent of

the number of criticality levels in the system. Observing that it is computationally expensive if we use the MM DBF-based test directly, we proposed a virtual deadline assignment algorithm which uses MM DBF as a complement to SM DBF to reduce the computation time further.

Using experimentation, we show that the MM DBF-based test strictly outperforms the SM DBF-based test GreedyTuning [25] for different settings. The proposed test also scales better than GreedyTuning with the number of criticality levels. However, the performance of the proposed test still drops as the number of criticality levels increases, especially for constrained deadline task sets. One possible reason for this performance drop is that there is not enough space to tune virtual deadline for each task in multi-criticality constrained deadline task systems.

The MM DBF is limited to constrained deadline mixed-criticality systems. Even though it seems to be straightforward to extend it to arbitrary deadline systems, it can become very pessimistic because there would exist more than one carry-over jobs. Therefore, another possible direction for future work is to address this problem and extend the MM DBF-based test to arbitrary deadline systems. This extension might either require some model changes to limit the number of carry-over jobs per task, or would need to derive an improved technique for bounding the demand of carry-over jobs.

Chapter 5

Component-based Technique for Supporting Low-Criticality Executions

In Chapter 4, we proposed a schedulability test for Vestal’s multi-criticality MC model under EDF-VD* scheduling policy. Although this test showed good improvement in schedulability performance, several concerns have been raised about the MC model itself as discussed in Section 1.3. One of the key concerns is that all low-criticality (LC) tasks are immediately suspended or offered degraded service even when only a single high-criticality (HC) task exceeds its allocated execution budget in LC mode (i.e., $C_i(L)$). Since such a severe penalty to LC tasks is impractical, there is a need for offering differentiated service to the LC tasks depending on which specific HC task(s) is requesting for additional resources. Such a task dependent penalty model seems reasonable, since these tasks generally have dependencies with each other (e.g., a set of tasks collectively implement a function in an application).

In this chapter, we propose a component-based technique to differentiate the services offered to LC tasks depending on the behavior of specific HC tasks. In this model, tasks in the system are assumed to be partitioned to different components. Instead of penalizing all LC tasks equally, when a single HC task exceeds $C_i(L)$, only LC tasks in the same component would be penalized and the budget of these LC tasks would be used to support the HC tasks in this component. However, if the number of HC tasks that execute beyond $C_i(L)$ in a component exceeds a predefined limit, all LC tasks in the system, including LC tasks in other components would be penalized. Thus, the impact of resource overload gradually progresses from within a component to across components. This model and technique has been previously published in [58].

We focus on dual-criticality MC systems scheduled on a uniprocessor platform for this work. In

Section 5.1, we present the motivation for the proposed technique. In Section 5.2, we introduce the component-based MC model for constrained deadline dual-criticality sporadic tasks. Considering EDF-VD* scheduling policy, we present a schedulability test based on MM DBF under a flat (non-hierarchical) scheduling framework in Section 5.3. In Section 5.4 we present a schedulability test under a hierarchical scheduling framework, which additionally supports *compositionality*¹. Finally, the capability of the proposed mechanism is evaluated in Section 5.5 in terms of schedulability and the ability to support LC jobs. In Table 5.1, we list the notations that are used in this chapter.

Table 5.1: Notations Used in Chapter 5

$C_i(L)$	WCET estimation of τ_i for LC mode.
$D_i(L)$	Virtual deadline of τ_i for LC mode.
$C_i(H)$	WCET estimation of τ_i for HC mode.
$J_{i,j}$	The j_{th} job of τ_i .
$r_{i,j}$	Release time of $J_{i,j}$.
t	An arbitrary time instant.
$t_E(\leq t)$	Time instant for an External Mode Switch (across components).
$t_I(\leq t_E)$	Time instant for an Internal Mode Switch (within component).
$TL \in \mathbb{N}$	Tolerance limit of a component.

5.1 Motivation and Contributions

In practice, most MC systems are component-based, wherein different vendors independently design and develop the various components (i.e., applications). Such a component-based design methodology is supported in automotive (e.g., AUTOSAR) as well as avionics (e.g., ARINC-653). Since we are interested in offering differentiated services to the LC tasks, it is then natural to consider the impact of HC task behavior within and across these components.

One extreme case is the reservation-based approach currently used in practice, which completely limits this impact to within components. Thus, the components are fully isolated from each other, so that LC task executions are unaffected by HC tasks in other components. However, as discussed in Chapter 1, such an approach results in severe under-utilization of the platform, because the reservations are often based on highly pessimistic WCET estimations.

On the other hand, many of the existing MC studies do not consider such component-based models at all. They either assume that all the LC tasks will be immediately suspended (e.g., [23, 24, 29, 31, 59])

¹Compositionality denotes the ability of a system to derive properties (e.g., schedulability) for higher-level components using derived properties of lower-level components.

or receive degraded service (e.g., [17, 19, 52, 60]) once a single HC task executes beyond its WCET estimation for LC mode. Hence, they are unable to control the service offered to LC tasks in a more fine-grained manner, that is, based on the behavior of specific HC tasks. Further, these studies also make a simplistic assumption that all HC tasks in the system will request for additional resources at the same time. This seems unreasonable, because in general such a dependence between tasks may not exist. For example, the probability that adaptive cruise control and anti-lock braking, both of which are critical, require additional resources at the same time is low because their execution time depends on different inputs. Cruise control would most likely require additional resources when the cameras and lidars provide dense data, whereas the execution of anti-lock braking mainly depends on the speed of the vehicle and the friction on the tyres.

To overcome the above limitations, we propose a component-based MC task model in this chapter. In this model, each component has a parameter called tolerance limit $TL \in \mathbb{N}$ denoting the maximum HC workload isolation limit of the component. Once a single HC task τ_i in a component $\mathbb{C}$ executes beyond $C_i(L)$, LC tasks in $\mathbb{C}$ are immediately suspended. However, as long as $\mathbb{C}$ does not violate its tolerance limit, that is, the number of HC tasks in $\mathbb{C}$ that exceed $C_i(L)$ at the same time is not more than TL , all other components are isolated from $\mathbb{C}$ and the LC tasks in them can continue to execute. We develop MM DBF-based schedulability tests for this model under EDF-VD* scheduling, and consider both a flat as well as hierarchical scheduling framework. Using experiments, we show that the proposed model significantly outperforms existing studies in terms of its ability to support the execution of LC tasks.

As discussed in Section 3.2.2 in Chapter 3, there are a couple of studies that offer differentiated service to the LC tasks [46, 47]. Neither of these consider a hierarchical scheduling framework, and hence they do not support compositionality. Further, as shown in the experiments in this chapter, the proposed component-based model and test has a much better schedulability performance than the model in [46], and this is mainly because the technique in [46] is limited to a restricted version of fixed-priority scheduling.

5.2 System Model

5.2.1 Task and Component

We focus on a constrained deadline dual-criticality sporadic task system τ scheduled on an uniprocessor platform (for more details about the dual-criticality model, refer to Section 2.2.2 in Chapter 2).

Thus, each task in τ would either be a low-criticality (LC) task or a high-criticality (HC) task. Tasks in the system are partitioned into *components*, where each component $\mathbb{C} = (\mathcal{W}, TL)$ comprises an *MC workload* $\mathcal{W}$ denoting a subset of tasks from τ and *HC Tolerance Limit* $TL \in \mathbb{N}$.

HC Tolerance Limit $TL \in \mathbb{N}$ denotes the maximum HC workload isolation limit of the component. As long as no more than TL HC tasks in the component simultaneously exceed their WCET estimations for LC mode, all job deadlines in the other components, including those of LC jobs, are still guaranteed to be met. Details about this parameter and the execution semantics of the model will be presented later. $\mathbb{C} = (\mathcal{W}, TL)$ is called a LC component if all the tasks in its workload are LC tasks, and for such components $TL = 0$. Otherwise, $\mathbb{C}$ is called an HC component.

Partitioning the task set into components is mainly driven by practical considerations. Since these components are developed independently, it is desirable to limit the impact of exceedance of WCET estimations within components as much as possible, while still guaranteeing the correctness of HC tasks. The HC tolerance limit TL of a component can achieve this goal. The value of TL could be set based on component properties if information about the runtime behavior of HC jobs is available, e.g., probability of execution requirement exceeding $C_i(L)$. It can also be determined such that the value of TL is maximized so as to support more LC task executions, while still maintaining system schedulability.

5.2.2 Execution Model

In this section, we define the execution semantics of a component $\mathbb{C} = (\mathcal{W}, TL)$. $\mathbb{C}$ has two execution modes, an *internal mode* that concerns the behavior of tasks in $\mathbb{C}$, and an *external mode* that concerns the behavior of tasks in the other components.

Internal Mode. Component $\mathbb{C}$ experiences *Internal Mode Switch* (or IMS for short) at the earliest time instant when any HC task in $\mathbb{C}$ executes beyond $C_i(L)$. The component switches its internal mode from LC to HC at this time instant. Prior to this mode switch, all the task deadlines are required to be met. After this switch however, all the LC tasks in $\mathbb{C}$ can be dropped, and only the HC task deadlines are required to be met. However, tasks in the other components in the system would not be impacted by IMS

External Mode. Component $\mathbb{C}$ experiences *External Mode Switch* (or EMS for short) at the earliest time instant when the $(TL + 1)^{st}$ HC task in $\mathbb{C}$ executes beyond $C_i(L)$. The component switches its external mode from LC to HC at this time instant. Prior to this mode switch, at most TL tasks in $\mathbb{C}$

can execute for up to $C_i(H)$ time units. After this switch however, all the HC tasks in $\mathbb{C}$ may execute up to $C_i(H)$ time units. Further, all the LC tasks in the system, including the LC tasks in the other components, are no longer required to meet deadlines.

Comparison with Mode-Switch in Vestal’s MC Model: The execution semantics of Vestal’s MC model is introduced in Section 2.2 in Chapter 2. In Vestal’s model, there is only one “system-wide” mode-switch triggered when a single HC task executes beyond $C_i(L)$. After this “system-wide” mode-switch, all LC tasks in the system could be ignored. In the proposed component-based MC model, component $\mathbb{C}$ has a “local” mode-switch (i.e., IMS) triggered when the first HC task executes beyond $C_i(L)$, and this mode-switch would only impact LC tasks within $\mathbb{C}$. Besides, $\mathbb{C}$ also has a “system-wide” mode-switch (i.e., EMS) after which all LC tasks in the system could be ignored. However, unlike Vestal’s model, EMS is only triggered when the $(TL + 1)^{st}$ HC task in $\mathbb{C}$ executes beyond $C_i(L)$.

If $\mathbb{C}$ is a LC component, then its internal and external modes are identical and equal to LC. On the other hand if $\mathbb{C}$ is an HC component, then IMS and EMS, together with the HC tolerance limit TL , are key mechanisms for supporting LC job executions.

If $TL > 0$, it is possible for IMS and EMS to occur at different time instants (asynchronously). Then, during the time interval when component $\mathbb{C}$ ’s internal mode is HC while its external mode is LC, other components are isolated from the internal mode switch of $\mathbb{C}$. That is, LC tasks in these components can continue their execution even though some HC tasks in $\mathbb{C}$ are already executing in HC mode.

Example 1 Here we use a simple example comprising two components to illustrate the concepts of the proposed model. $\mathbb{C}_L = (\mathcal{W}_L, TL_L = 0)$ is a LC component and $\mathcal{W}_L$ contains only one LC task τ_1 . $\mathbb{C}_H = (\mathcal{W}_H, TL_H = 1)$ is an HC component and $\mathcal{W}_H$ contains two HC tasks, τ_2 and τ_3 , and one LC task τ_4 . Since $TL_H = 1$, component $\mathbb{C}_H$ ensures low criticality isolation for component $\mathbb{C}_L$ at all those time instants when at most 1 HC task in $\mathbb{C}_H$ is executing in HC mode. In particular, when the first HC task (τ_2 or τ_3) in $\mathbb{C}_H$ requests execution in excess of $C_i(L)$, $\mathbb{C}_H$ experiences IMS. After IMS, LC task τ_4 in $\mathbb{C}_H$ can be dropped because it is no longer required to meet deadlines, while LC task τ_1 in $\mathbb{C}_L$ must continue to execute as before. When the second HC task in $\mathbb{C}_H$ requests execution in excess of $C_i(L)$, isolation limit $TL_H = 1$ is violated and $\mathbb{C}_H$ experiences EMS, and thereafter task τ_1 in $\mathbb{C}_L$ cannot be guaranteed to be scheduled and may be dropped.

Scheduling Strategy. We apply EDF-VD* scheduling policy in the component-based MC model and assume that LC tasks are not considered for scheduling once it becomes known that their deadlines

are not required to be met. Note that an HC task in component $\mathbb{C}$ that executes in LC mode after IMS of $\mathbb{C}$ will continue to be scheduled using its virtual deadline $D_i(L)$, unless it switches to HC mode. However, after EMS of $\mathbb{C}$, all the HC tasks are assumed to switch to HC mode and will be scheduled using their actual deadlines. Below, we use a simple example to illustrate how EDF-VD* is applied in this model.

Example 2 Suppose $\mathbb{C}$ with $TL = 1$ has three HC task $\{\tau_1, \tau_2, \tau_3\}$. At t_I , τ_1 executes beyond $C_1(L)$ and triggers IMS. Thereafter, τ_1 switches to its actual deadline D_1 , but τ_2 and τ_3 are still scheduled using their virtual deadlines. At t_E , EMS is triggered after τ_2 executes for $C_2(L)$ time units without signaling completion. After t_E , both τ_2 and τ_3 would revert to their actual deadlines.

We consider two different scheduling frameworks; a flat (non-hierarchical) framework in which all tasks in all components are collectively scheduled by a single scheduler, and a hierarchical framework in which the tasks in components are scheduled by intra-component schedulers and the components themselves are scheduled by an inter-component scheduler. The flat framework is relevant in applications that do not use hierarchical scheduling (e.g., Deos Real-Time Operating System for avionics [61]), whereas the hierarchical framework is relevant in applications that require compositionality (e.g., ARINC653 in avionics [8]). However, under the hierarchical scheduling framework, the elements of workload of a component can either be regular MC tasks or child components which are represented by interfaces.

5.3 Schedulability Test for Flat Scheduling Framework

Let $[0, t]$ denote the time interval of interest. Suppose the External Mode Switch or EMS of $\mathbb{C}$ occurs at $t_E (\leq t)$ and the Internal Mode Switch or IMS of $\mathbb{C}$ occurs at $t_I (\leq t_E)$. If $\mathbb{C}$ is a LC component, then it has no IMS or EMS, and tasks within it will be dropped after the earliest EMS of any component in the system.

For an HC task τ_i in the workload of $\mathbb{C}$, let t_i denote the time instant after which jobs of τ_i would be allowed to execute upto $C_i(H)$ time units. By definition $t_I \leq t_i \leq t_E$. For a LC task τ_i in the workload of $\mathbb{C}$, let t_i denote the time instant when it is dropped. In this case, t_i is either equal to t_I or the EMS of any HC component, whichever is earlier.

In the rest of this section, we derive a schedulability test for the component-based MC model under a flat scheduling framework based on the MM DBF derived in Section 4.3 in Chapter 4.

5.3.1 Demand of Jobs

In Section 4.3.1 of Chapter 4, we defined four types of jobs $J_{i,X}$, where $X \in \{A, B, C, D\}$, that experience mode-switches. Jobs $J_{i,A}$ and $J_{i,C}$ experienced mode-switch at $t^*(m-1) \equiv 0$ (previous mode-switch), and jobs $J_{i,B}$ and $J_{i,D}$ experienced mode-switch at $t^*(m)$ (mode-switch under analysis). However, in a dual-criticality system, there is only one mode-switch and hence jobs $J_{i,A}$ and $J_{i,C}$ do not exist. Thus, we only need to consider jobs $J_{i,B}$ and $J_{i,D}$ defined as follows.

- $J_{i,B}$: $r_{i,B} \leq t_i \wedge r_{i,B} + T_i > t_i \wedge L_i = LC$
- $J_{i,D}$: $r_{i,D} \leq t_i \wedge r_{i,D} + T_i > t_i \wedge L_i = HC$

Let $dbf_{MM}(J_i, t_i, t)$ denote the DBF of job J_i for a given time interval length t and time instant t_i . Note here t_i denotes the time instant when the job either switches its execution mode or gets dropped. Then, based on the discussions in Section 4.3.1 of Chapter 4, the DBF for jobs $J_{i,B}$ and $J_{i,D}$ can be defined as follows. Note, these equations are identical to Equations 4.5 and 4.7 in Section 4.3.1, and can be obtained by substituting $t^*(m) = t_i$.

$$dbf_{MM}(J_{i,B}, t_i, t) = \begin{cases} \min \{t_i - r_{i,B}, C_i(L)\}, & r_{i,B} + D_i \leq t \\ 0, & \text{otherwise} \end{cases} \quad (5.1)$$

$$dbf_{MM}(J_{i,D}, t_i, t) = \begin{cases} C_i(L), & r_{i,D} + D_i(L) < t_i \\ C_i(H), & r_{i,D} + D_i(L) \geq t_i \\ & \text{and } r_{i,D} + D_i \leq t \\ \min \{t_i - r_{i,D}, C_i(L)\}, & t_i \leq r_{i,D} + D_i(L) \leq t \\ & \text{and } r_{i,D} + D_i > t \\ 0, & r_{i,D} + D_i(L) > t \end{cases} \quad (5.2)$$

5.3.2 DBF of Task τ_i

In this section, we define the MM DBF for a single task τ_i in a dual-criticality system. Note, the DBF equations presented here are equivalent to Equations 4.9 and 4.11 presented in Section 4.3.2 of Chapter 4, when they are restricted to dual-criticality systems. We define $dbf_{MM}(\tau_i, t_i, t)$ using four

sub-cases $dbf_{MM}^x(\tau_i, t_i, t)$, where $x \in \{a, b, c, d\}$ as follows.

a: $L_i = LC$:

$$dbf_{MM}^a(\tau_i, t_i, t) = \left\lfloor \frac{t_i}{T_i} \right\rfloor C_i(L) + dbf_{MM}(J_{i,B}, t_i, t) \quad (5.3)$$

b: $L_i = HC$ and $t - t_i < D_i - D_i(L)$:

$$dbf_{MM}^b(\tau_i, t_i, t) = \left\lfloor \frac{t_i}{T_i} \right\rfloor C_i(L) + dbf_{MM}(J_{i,D}, t_i, t) \quad (5.4)$$

c: $L_i = HC$ and $t - t_i \geq D_i$:

$$\begin{aligned} dbf_{MM}^c(\tau_i, t_i, t) &= n_L C_i(L) + dbf_{MM}(J_{i,D}, t_i, t) + n_H C_i(H), \text{ where} \\ n_L &= \left\lfloor \frac{t_i - r_{i,1}}{T_i} \right\rfloor = \left\lfloor \frac{t - D_i}{T_i} \right\rfloor - \left\lceil \frac{t - D_i - t_i}{T_i} \right\rceil, \text{ and} \\ n_H &= \left\lfloor \frac{t - D_i}{T_i} \right\rfloor - n_L. \end{aligned} \quad (5.5)$$

d: $L_i = HC$ and $D_i - D_i(L) \leq t - t_i < D_i$:

$$dbf_{MM}^d(\tau_i, t_i, t) = \max \left\{ dbf_{MM}^b(\tau_i, t_i, t), dbf_{MM}^c(\tau_i, t_i, t) \right\} \quad (5.6)$$

5.3.3 DBF of Component $\mathbb{C}$

In this section, we derive the DBF for a component. Let $dbf_{MM}(\mathbb{C}, t_I, t_E, t)$ denote the DBF of component $\mathbb{C} = \{\mathcal{W}, TL\}$ given a time interval length t and mode-switch instants t_I (IMS) and t_E (EMS). We first consider the case when $\mathbb{C}$ has $TL = 0$ and extend it to the case when $TL > 0$. Note that among all the HC tasks in $\mathbb{C}$, at most TL of them can execute upto $C_i(H)$ time units during $[t_I, t_E]$, while all the remaining HC tasks can only execute upto $C_i(H)$ time units after t_E .

If $TL = 0$, then this means $\mathbb{C}$'s internal and external mode-switch would occur at the same time, i.e., $t_I = t_E$. Thus, each HC task τ_i in $\mathbb{C}$ will switch to HC mode at $t_i = t_I = t_E$. In this case, $dbf(\mathbb{C}, t_I, t_E = t_I, t)$ is given as follows.

$$dbf_{MM}(\mathbb{C}, t_I, t_E = t_I, t) = \sum_{\tau_i \in \mathbb{C}} dbf_{MM}(\tau_i, t_I, t) \quad (TL = 0) \quad (5.7)$$

Otherwise, if $TL > 0$, at most TL HC tasks can execute upto $C_i(H)$ during $[t_I, t_E]$. Thus, we then need to determine which HC tasks could request upto $C_i(H)$ units of budget during $[t_I, t_E]$ so as to maximize

the total demand. The following lemma asserts that for any HC task, its demand is maximized when it switches to HC mode either at t_I or t_E .

Lemma 5 *If a HC task τ_i switches to HC mode at some time $t_i \in [t_I, t_E]$, then $dbf_{MM}(\tau_i, t_i, t)$ is maximized when t_i is either equal to t_E or t_I .*

Proof: Suppose τ_i satisfies condition b when $t_i = t_E$, i.e., $t - t_E < D_i - D_i(L)$. Then as t_i decreases, τ_i could eventually satisfy condition d, i.e., $D_i - D_i(L) \leq t - t_i < D_i$, and finally condition c, i.e., $t - t_i \geq D_i$. Without loss of generality, assume that τ_i satisfies condition b for $t_i \in (t_b, t_E]$, condition d for $t_i \in (t_d, t_b]$, and condition c for $t_i \in [t_I, t_d]$, where $t_I \leq t_d \leq t_b \leq t_E$.

Case 1 ($t_i \in [t_I, t_d]$): In this case, $dbf_{MM}^c(\tau_i, t_i, t)$ (see Equation 5.5) is maximized if $t_i = t_I$. This is because as t_i decreases from t_d to t_I , the number of jobs generating $C_i(H)$ demand will remain the same or increase, while the total number of jobs that generate demand for this time interval remains unchanged.

Case 2 ($t_i \in (t_b, t_E]$): In this case,

$$dbf_{MM}^b(\tau_i, t_i, t) = \left\lfloor \frac{t_i}{T_i} \right\rfloor C_i(L) + dbf_{MM}(J_{i,D}, t_i, t).$$

Then as t_i increases from t_b to t_E , $dbf_{MM}(J_{i,D}, t_i, t)$ and $\left\lfloor \frac{t_i}{T_i} \right\rfloor \times C_i(L)$ will stay the same or increase. Thus $dbf_{MM}^b(\tau_i, t_i, t)$ is maximized when $t_i = t_E$.

Case 3 ($t_i \in (t_d, t_b]$): We know that

$$dbf_{MM}^d(\tau_i, t_i, t) = \max \left\{ dbf_{MM}^b(\tau_i, t_i, t), dbf_{MM}^c(\tau_i, t_i, t) \right\}.$$

While $dbf_{MM}^b(\tau_i, t_i, t) | t_i \in (t_d, t_b]$ is maximized if $t_i = t_b$, $dbf_{MM}^c(\tau_i, t_i, t) | t_i \in (t_d, t_b]$ stays the same. Since $dbf_{MM}^b(\tau_i, t_b, t) \leq dbf_{MM}^b(\tau_i, t_E, t)$ and $dbf_{MM}^c(\tau_i, t_b, t) \leq dbf_{MM}^c(\tau_i, t_I, t)$, combining the above three cases, we conclude that $dbf_{MM}(\tau_i, t_i, t)$ is maximized when $t_i = t_I$ or $t_i = t_E$. ■

From Lemma 5, we know that the demand of an HC task τ_i is maximized if it switches to HC mode at either $t_i = t_E$ or $t_i = t_I$. Let

$$\Delta_i = \max\{0, dbf_{MM}(\tau_i, t_I, t) - dbf_{MM}(\tau_i, t_E, t)\}.$$

Here Δ_i denotes the maximum possible increase in the demand of τ_i (if it increases) for a time interval length t if τ_i is chosen as one of the TL tasks to switch to HC mode before t_E . Once we compute Δ_i

for all the HC tasks in component $\mathbb{C}$, we sort these values in descending order and select the first TL elements. Let $\mathcal{G}$ denote the TL chosen HC tasks, then the total demand of $\mathbb{C}$ for a time interval length t can be computed as follows.

$$dbf_{MM}(\mathbb{C}, t_I, t_E, t) = \sum_{L_i=HC} dbf_{MM}(\tau_i, t_E, t) + \sum_{\tau_i \in \mathcal{G}} \Delta_i + \sum_{L_i=LC} dbf_{MM}(\tau_i, t_I, t) \quad (5.8)$$

5.3.4 DBF Optimization

In this section, we apply the optimization technique in Theorem 4 of Chapter 4 to further bound the execution demand of component $\mathbb{C}$ for the case $t_E < t$. First, we split $dbf_{MM}(\tau_i, t_I, t)$ (for LC task) and $dbf_{MM}(\tau_i, t_E, t)$ (for HC task) into two parts, where $EXPH_i$ denotes the maximum possible demand of τ_i for the interval $[t_E, t)$, and $EXPL_i$ denotes the remaining demand of τ_i for the interval $[0, t_E)$. Since LC task would be ignored at $t_I (\leq t_E)$, $dbf_{MM}(\tau_i, t_I, t)$ can be split as follows.

$$EXPL_i = dbf_{MM}(\tau_i, t_I, t)$$

$$EXPH_i = 0$$

Note that, by definition, the SM DBF $dbf_{SM}(\tau_i, t - t_E, 2)$ (Equation 4.3 in Section 4.2) upper bounds the execution demand of task τ_i for a time interval length $t - t_E$ in HC mode. Thus, the execution demand of τ_i during $[t_E, t]$ cannot exceed $dbf_{SM}(\tau_i, t - t_E, 2)$. As a result, for a HC task, $dbf_{MM}(\tau_i, t_E, t)$ can be split as follows.

$$EXPL_i = dbf_{MM}(\tau_i, t_E, t) - dbf_{SM}(\tau_i, t - t_E, 2)$$

$$EXPH_i = dbf_{SM}(\tau_i, t - t_E, 2)$$

Since the first deadline miss is assumed to happen at time instant $t (> t_E)$, the demand before t_E cannot exceed t_E . Otherwise, the first deadline miss would happen at or before t_E . Thus the total demand during $[0, t_E)$ can be bounded by t_E . Based on this key observation, $dbf_{MM}(\mathbb{C}, t_I, t_E, t)$ can

be more tightly bounded as follows.

$$\begin{aligned}
 dbf_{MM}(\mathbb{C}, t_I, t_E, t) &= \text{EXPL} + \text{EXPH} + \sum_{\Delta_i \in \mathcal{G}} \Delta_i, \text{ where} \\
 \text{EXPH} &= \sum_{L_i = HC} dbf_{SM}(\tau_i, t - t_E, 2), \text{ and} \\
 \text{EXPL} &= \min \left\{ t_E, \sum_{L_i = LC} dbf_{MM}(\tau_i, t_I, t) + \sum_{L_i = HC} (dbf_{MM}(\tau_i, t_E, t) - dbf_{SM}(\tau_i, t - t_E, 2)) \right\}
 \end{aligned} \tag{5.9}$$

5.3.5 Schedulability Test and Tolerance Limit

In this section, we derive the schedulability test for a MC system comprising p HC components $\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_p$ and q LC components $\mathbb{C}_{p+1}, \mathbb{C}_{p+2}, \dots, \mathbb{C}_{p+q}$, scheduled under a flat scheduling framework.

Each HC component $\mathbb{C}_i$ can switch its internal mode to HC independently at t_{Ii} . Once the first HC component switches its external mode to HC at t_E , all the LC tasks in the system are immediately dropped, and all the HC tasks in the system can thereafter execute upto $C_i(H)$ time units.

Suppose the first deadline miss in the system occurs at some time instant t . Then, the total maximum demand generated by the system in $[0, t]$ must be greater than t . This assertion immediately leads to the following theorem that presents the schedulability test.

Theorem 6 *A mixed-criticality system comprising p HC components and q LC components is schedulable under a flat scheduling framework if, $\forall t : 0 \leq t \leq t^{max}, \forall t_E : 0 \leq t_E \leq t, \forall t_{Ii} : 0 \leq t_{Ii} \leq t_E$,*

$$\sum_{i=1}^{i \leq p+q} dbf_{MM}(\mathbb{C}_i, t_{Ii}, t_E, t) \leq t, \tag{5.10}$$

where t^{max} is a pseudo-polynomial in the size of the input (derived in Section 5.3.6).

The complexity of the schedulability test in Theorem 6 is exponential in the number of HC components, because we need to consider a separate internal mode switch instant for each component. In practice however, we expect the number of HC components scheduled on a single processor to be relatively small, and then the complexity of the proposed test is pseudo-polynomial in the size of the input. Besides, if there is freedom to select the allocation of system tasks to components, then it is feasible to create a component structure comprising only two components, while still fully supporting LC task executions. All the HC tasks in the system are allocated to a single HC component $\mathbb{C}_H = \{\mathcal{W}_H, TL_H\}$,

and each LC task can be either allocated to $\mathbb{C}_H$ or to a LC component $\mathbb{C}_L = \{\mathcal{W}_L, TL_L = 0\}$. This two-component system is sufficient to consider all the possible design choices for isolating HC and LC task executions. This can be done by considering different values for the tolerance limit TL_H , and by considering different allocations of LC tasks to the two components. We can choose the maximum possible value for TL_H as long as the resulting system is still schedulable. Higher tolerance limit indicates support for more LC task executions, and thus better resource utilization. In Section 5.5, we show through simulations that the proposed component-based technique outperforms existing studies even with this two-component structure. However, if the allocation of tasks to components is fixed and the number of HC components is not small, then the hierarchical scheduling framework can be used to reduce the complexity of the test.

5.3.6 Upper Bound of t^{max}

Since the execution demand of all components in the system cannot be bounded by the MM DBF derived in Chapter 4, the bound of t^{max} for Theorem 5 in Chapter 4 cannot be directly used for the test in Theorem 6. In this section, we derive a new upper bound for t^{max} for the schedulability test in Theorem 6. Consider a MC system with p HC components $\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_p$ and q LC components $\mathbb{C}_{p+1}, \mathbb{C}_{p+2}, \dots, \mathbb{C}_{p+q}$. Suppose there exists a t_F such that $\sum_{i=1}^{i \leq p+q} dbf_{MM}(\mathbb{C}_i, t_{Ii}, t_E, t) > t_F$. Then it must be that

$$\begin{aligned} & \sum_{i=1}^{i \leq p+q} \left(\sum_{L_j=HC} dbf_{MM}(\tau_j, t_E, t_F) \right) + \sum_{i=1}^{i \leq p+q} \left(\sum_{\tau_j \in \mathcal{G}_i} \max\{0, dbf_{MM}(\tau_j, t_{Ii}, t) - dbf_{MM}(\tau_j, t_E, t)\} \right) \\ & + \sum_{i=1}^{i \leq p+q} \left(\sum_{L_j=LC} dbf_{MM}(\tau_j, t_{Ii}, t_F) \right) > t_F \end{aligned}$$

Let A_i denote the set of HC tasks in $\mathbb{C}_i$ that belong to $\mathcal{G}_i$ with $dbf_{MM}(\tau_j, t_{Ii}, t) - dbf_{MM}(\tau_j, t_E, t) > 0$ and B_i denote the rest of HC tasks in $\mathbb{C}_i$, where $|A_i| \leq TL_i$. Let

- $U_{A_i}(L) = \sum_{\tau_j \in A_i} u_j(L)$
- $U_{A_i}(H) = \sum_{\tau_j \in A_i} u_j(H)$
- $U_{B_i}(L) = \sum_{\tau_j \in B_i} u_j(L)$
- $U_{B_i}(H) = \sum_{\tau_j \in B_i} u_j(H)$

- $U_{h_i}(L) = \sum_{L_j=HC \wedge \tau_j \in \mathbb{C}_i} u_j(L) = U_{B_i}(L) + U_{A_i}(L)$
- $U_{h_i}(H) = \sum_{L_j=HC \wedge \tau_j \in \mathbb{C}_i} u_j(H) = U_{B_i}(H) + U_{A_i}(H)$
- $U_{l_i}(L) = \sum_{L_j=LC \wedge \tau_j \in \mathbb{C}_i} u_j(L)$
- $t_I^{MIN} = \min_{i=1}^{p+q} \{t_{I_i}\}$

Then, we have

$$\begin{aligned}
& \sum_{i=1}^{i \leq p+q} \left(\sum_{\tau_j \in A_i} dbf_{MM}(\tau_j, t_{I_i}, t_F) + \sum_{\tau_j \in B_i} dbf_{MM}(\tau_j, t_E, t_F) + \sum_{L_j=LC} dbf_{MM}(\tau_j, t_{I_i}, t_F) \right) > t_F \Rightarrow \\
& \sum_{i=1}^{i \leq p+q} \left(\underbrace{\sum_{\tau_j \in A_i} (t_I^{MIN} u_j(L) + 2C_j(H) + (t_F - t_I^{MIN}) u_j(H))}_{\geq \sum_{\tau_j \in A_i} dbf_{MM}(\tau_j, t_{I_i}, t_F)} \right) \\
& + \sum_{i=1}^{i \leq p+q} \left(\underbrace{\sum_{\tau_j \in B_i} (t_E u_j(L) + 2C_j(H) + (t_F - t_E) u_j(H))}_{\geq \sum_{\tau_j \in B_i} dbf_{MM}(\tau_j, t_E, t_F)} \right) + \sum_{i=1}^{i \leq p+q} \left(\underbrace{\sum_{L_j=LC} (t_E u_j(L) + C_j(L))}_{\geq \sum_{L_j=LC} dbf_{MM}(\tau_j, t_{I_i}, t_F)} \right) > t_F \\
& \Rightarrow \sum_{i=1}^{i \leq p+q} \left(t_F U_{A_i}(H) + \overbrace{t_I^{MIN} (U_{A_i}(L) - U_{A_i}(H))}^{\text{maximizes when } t_I^{MIN}=0} \right) + \sum_{i=1}^{i \leq p+q} (t_F U_{B_i}(H) + t_E (U_{B_i}(L) - U_{B_i}(H))) \\
& + \left(\sum_{i=1}^{i \leq p+q} t_E U_{l_i}(L) \right) + \sum_{L_j=HC} 2C_j(H) + \sum_{L_j=LC} C_j(L) > t_F \\
& \Rightarrow \underbrace{\sum_{i=1}^{i \leq p+q} \left(t_F \left(\overbrace{U_{A_i}(H) + U_{B_i}(H)}^{=U_{h_i}(H)} \right) + t_E \left(U_{l_i}(L) + \underbrace{U_{B_i}(L) - U_{B_i}(H)}_{=U_{h_i}(L) - U_{A_i}(L) + U_{A_i}(H) - U_{h_i}(H)} \right) \right)}_{EXP} \\
& + \sum_{L_j=HC} 2C_j(H) + \sum_{L_j=LC} C_j(L) > t_F
\end{aligned}$$

To maximize EXP , A_i is the set of HC tasks with TL_i highest values for $u_j(H) - u_j(L)$. If $U_{l_i}(L) + U_{B_i}(L) - U_{B_i}(H) < 0$, EXP is maximized when $t_E = 0$. Otherwise if $U_{l_i}(L) + U_{B_i}(L) - U_{B_i}(H) \geq 0$, EXP is maximized when $t_E = t_F$. Therefore, we can use 0 or t_F to replace t_E in EXP according to the above two cases to get the upper bound for t_F .

5.4 Schedulability Test for Hierarchical Scheduling Framework

Hierarchical scheduling has emerged as an effective mechanism to support temporal partitioning between applications, serving as a common scheduling paradigm in many MC systems in practice [8]. It is preferred in practice because it supports *compositionality* so that higher-level properties can be derived from verified component-level properties. To increase the practical relevance of the proposed component-based MC model, we also derive a schedulability test for hierarchical scheduling framework. The schedulability test for hierarchical scheduling framework has lower complexity than the one for flat scheduling framework.

5.4.1 Execution Strategy under Hierarchical Scheduling

Under a hierarchical scheduling framework, each component will have an additional parameter S denoting its local scheduler, and hence a component can be specified as $\mathbb{C} = (\mathcal{W}, TL, S)$. The component workload $\mathcal{W}$ comprises regular MC tasks as well as interface tasks that represent the child components. The tasks in the workload $\mathcal{W}$ are scheduled by the local scheduler S , independently of all the other components in the system.

Component interfaces have been widely used in traditional hierarchical systems to abstractly represent the resource demand and supply of components (see for example [62]). From the component's perspective, its interface represents the resource demand of its workload. While from the perspective of its parent component or system, the interface represents the resource supply that the parent guarantees. These interfaces of components are essential for satisfying the property of compositionality.

Resource models such as periodic have been previously defined as interfaces for components in traditional hierarchical systems [62]. Analogously, we now propose a mixed-criticality periodic resource (MCPR) model for mixed-criticality components. Since we focus on dual-criticality systems, we assume that the MCPR model can have at most two criticality levels.

Definition 12 A *Mixed-Criticality Periodic Resource (MCPR)* is defined as $\mathbb{I} = (T, L, \mathcal{C})$, where T denotes the period, $L \in \{LC, HC\}$ denotes the criticality level, and $\mathcal{C} = \{C(L), C(H)\}$ is a list of resource capacities. $C(L)$ denotes LC resource capacity and $C(H)$ denotes HC resource capacity.

A component $\mathbb{C}$ can be abstracted as an MCPR interface $\mathbb{I} = (T, L, \mathcal{C})$, and the corresponding interface task $(T, L, \mathcal{C}, T)$ represents $\mathbb{C}$ in the workload of its parent component. The period T of this interface is assumed to be already specified by the system designer as in the standard literature on hierarchical scheduling (e.g., see [62]). For instance, this period could be determined based on either component-level requirements or considerations for overheads such as context-switches. The criticality level L of the interface is equal to the criticality level of the component it is representing.

Mode of the interface. The semantics of interface $\mathbb{I}$ (and the corresponding interface task) depend on its *criticality mode* at runtime, which in turn depends on the criticality mode of component $\mathbb{C}$. In fact, the criticality mode of $\mathbb{I}$ is identical to the external mode of $\mathbb{C}$. When $\mathbb{C}$ experiences EMS, the mode of the interface and interface task switches from LC to HC. While the interface is in LC mode, it is guaranteed to request no more than $C(L)$ time units of resource periodically every T time units from the parent component. However, once it switches to HC mode, it can thereafter request up to $C(H)$ time units of resource every T time units. Below we use a simple example to illustrate how the resource supplied to a interface changes after EMS occurs in the system.

Example 3 Suppose a system has an HC component $\mathbb{C}_H$ with interface

$$\mathbb{I}_H = (T_H, HC, \mathcal{C}_H = \{C_H(L), C_H(H)\})$$

and a LC component $\mathbb{C}_L$ with interface

$$\mathbb{I}_L = (T_L, LC, \mathcal{C}_L = \{C_L(L)\}).$$

Initially, $\mathbb{I}_H$ could request upto $C_H(L)$ units of resource every T_H time units and $\mathbb{I}_L$ could request upto $C_L(L)$ units of budget every T_L time units. After EMS occurs in $\mathbb{C}_H$, LC tasks in $\mathbb{C}_H$ could be ignored but the resource allocated to both $\mathbb{I}_H$ and $\mathbb{I}_L$ would not change. However, once EMS occurs, $\mathbb{I}_H$ could request upto $C_H(H)$ time units of resource every T_H time units thereafter, but $\mathbb{I}_L$ could no longer request any resource.

Figure 5.1: MCPR worst-case resource supply pattern A

Figure 5.2: Boundary case for MCPR worst-case resource supply pattern A

5.4.2 MCPR Supply Bound Function

The supply bound function (sbf) of a resource model characterizes the minimum resource supply guaranteed by the model to the underlying component. In this section, we derive the sbf for a MCPR interface $\mathbb{I} = (T, L, C)$ of a component $\mathbb{C} = (\mathcal{W}, TL, S)$. Let $sbf_{\mathbb{I}}(t_E, t)$ denote the sbf for a time interval of length t , where $t_E (\leq t)$ denotes the time instant for EMS of component $\mathbb{C}$. As the resource is supplied periodically, component $\mathbb{C}$ is guaranteed to receive either $C(L)$ or $C(H)$ units of resource every T time units in LC or HC mode, respectively. We use the following additional notations in this section.

- s_1 denotes the start time of the first interface period within time interval $[0, t)$.
- n denotes the number of interface periods within interval $[0, t)$.
- n_E denotes the number of interface periods within interval $[0, t_E)$.
- s_E denotes the start of an interface period that experiences EMS, i.e., $s_E \leq t_E < e_E$, where $e_E = s_E + T$.

- e denotes the start of interface period after t , i.e., $e = s_1 + n \times T + T$.
- For simplicity of presentation, we also use the short-cut notation $[x]_0 = \max\{0, x\}$.

When $t_E = t$, there is no external mode switch for component $\mathbb{C}$ in the interval of interest, and the component and interface are only executing in LC mode. Therefore, $sbf_{\mathbb{I}}(t_E, t)$ in this case is identical to the sbf defined for periodic resource models with $\mathbb{I}$ supplying $C(L)$ units of resource periodically [62]. Thus, in this case, minimal resource is supplied when $s_1 = T - C(L)$ and $n = \left\lceil \left\lfloor \frac{t - (T - C(L))}{T} \right\rfloor \right\rceil$. We record this sbf in the following equation.

$$sbf_{\mathbb{I}}(t_E, t) = n \times C(L) + [t - 2(T - C(L)) - n \times T]_0 \quad \text{If } t_E = t \quad (5.11)$$

For the case when $t_E < t$, there are two possible resource supply patterns, denoted A and B, that can lead to the minimum resource supply. We now present these two patterns and the corresponding sbf equations, $sbf_{\mathbb{I}}(t_E, t)_{[A]}$ and $sbf_{\mathbb{I}}(t_E, t)_{[B]}$.

Figure 5.3: MCPR worst-case resource supply pattern B

Pattern A: $s_1 = T - C(L)$. The scenario of pattern A is shown in Figure 5.1, where $n_E = \left\lceil \left\lfloor \frac{t_E - (T - C(L))}{T} \right\rfloor \right\rceil$ and $n = \left\lceil \left\lfloor \frac{t - (T - C(L))}{T} \right\rfloor \right\rceil$. In the first period, $C(L)$ units of resource are supplied as early as possible and hence during $[0, 2 \times (T - C(L))]$, no resource is supplied. In the following periods until time instant $s_E (= n_E \times T + T - C(L))$, $C(L)$ units are supplied as late as possible. In the period $[s_E, e_E]$, the amount of supply depends on the distance of t_E from s_E . If $t_E - s_E < C(L)$, then the resource supply in this period cannot be exhausted when component $\mathbb{C}$ has EMS at t_E . Therefore interface $\mathbb{I}$ will provide $C(H)$ units of resource in this period, because it can signal its mode switch to the parent component. On the other hand, if $t_E - s_E \geq C(L)$ as in the example figure, then the resource supply in $[s_E, e_E]$ can be exhausted before component $\mathbb{C}$ experiences EMS, and hence the interface may only provide $C(L)$ units in this period. After time instant e_E , the interface is guaranteed to provide $C(H)$ units of resource in every period. An important boundary case to consider is when $e = n \times T + 2T - C(L) = e_E$ and $t_E - s_E \geq C(L)$. That is, when t_E and t are in the same period and the interface can exhaust its resource supply before EMS of component $\mathbb{C}$ (scenario shown

in Figure 5.2). In this case, the minimum supply in this period can happen when it is provided as late as possible (for instance when $e - t > C(H) - C(L)$). We record the *sb**f* corresponding to the pattern of Figures 5.1 and 5.2 below.

$$sbf_{\mathbb{I}}(t_E, t)_{[A]} = \begin{cases} n_E \times C(L) + (n - n_E) \times C(H) \\ + [t - (2T - C(L) - C(H)) - n \times T]_0 & t_E - s_E < C(L) \\ \\ (n_E + 1) \times C(L) + (n - n_E - 1) \times C(H) \\ + [t - (2T - C(L) - C(H)) - n \times T]_0 & e \neq e_E \wedge \\ & t_E - s_E \geq C(L) \\ \\ n_E \times C(L) \\ + \min \{C(L), [t - (2T - C(L) - C(H)) - n \times T]_0\} & e = e_E \wedge \\ & t_E - s_E \geq C(L) \end{cases} \quad (5.12)$$

Pattern B: $s_1 = T - C(L) - (x_E - t_E)$, where $x_E = \lceil \frac{t_E}{T} \rceil \times T$. Scenario of pattern B is shown in Figure 5.3, which is obtained by shifting pattern A in Figure 5.1 by $x_E - t_E$. In this case,

$$n_E = \left\lfloor \left\lfloor \frac{t_E - s_1}{T} \right\rfloor \right\rfloor_0, n = \left\lfloor \left\lfloor \frac{t - s_1}{T} \right\rfloor \right\rfloor_0, e_E = t_E - C(L) + T \\ \text{and } e = n \times T + T + s_1.$$

The *sb**f* corresponding to this shifted supply pattern is given below. It is similar to the previous case, except that the interface period containing t_E is now guaranteed to supply no more than $C(L)$ time units.

$$sbf_{\mathbb{I}}(t_E, t)_{[B]} = \begin{cases} (n_E + 1) \times C(L) + (n - n_E - 1) \times C(H) \\ + [t - s_1 - (T - C(H)) - n \times T]_0 & e \neq e_E \\ \\ n_E \times C(L) \\ + \min \{C(L), [t - s_1 - (T - C(H)) - n \times T]_0\} & e = e_E \end{cases} \quad (5.13)$$

The following lemma proves that it is sufficient to consider the above two supply patterns for determining the $sb f$.

Lemma 6 *When $t_E < t$, pattern A or B are the only two possible supply patterns that can result in the minimal resource supply from interface $\mathbb{I}$.*

Proof: Suppose there exists a $s_1 \in [0, T)$ such that $s_1 \neq T - C(L)$ (pattern A) and $s_1 \neq T - C(L) - (x_E - t_E)$ (pattern B), but s_1 leads to the minimal supply pattern for time interval length t .

Case 1 ($s_1 = T - C(L) + \epsilon | 0 < \epsilon \leq C(L)$): In this case, it is easy to see that the supply will be greater than or equal to $sb f_{\mathbb{I}}(t_E, t)_{[A]}$, because the supply for the first interface period will increase by ϵ and the supply for the last interface period will decrease by at most ϵ .

Case 2 ($s_1 = T - C(L) - (x_E - t_E) + \epsilon | 0 < \epsilon < (x_E - t_E)$): In this case, the supply for the interface period containing t_E will stay the same or increase by $C(H) - C(L)$ while the supply for the last interface period may decrease by at most ϵ compared with the case when $s_1 = T - C(L) - (x_E - t_E)$. Therefore this supply is also minimized when $\epsilon \rightarrow x_E - t_E$ or $\epsilon \rightarrow 0$.

Case 3 ($s_1 = T - C(L) - (x_E - t_E) - \epsilon | 0 < \epsilon \leq T - C(L) - (x_E - t_E)$): In this case, the supply for the interface period containing t_E will stay the same, while the supply for the last interface period may stay the same or increase compared with the case when $s_1 = T - C(L) - (x_E - t_E)$. Therefore in this case as well, the supply is minimized when $\epsilon \rightarrow 0$. Combining the above cases, we can conclude that the supply is minimized with either pattern A or pattern B. ■

Thus, a safe lower bound for $sb f_{\mathbb{I}}$ for the case when $t_E < t$ can be stated as follows.

$$sb f_{\mathbb{I}}(t_E, t) = \min \{ sb f_{\mathbb{I}}(t_E, t)_{[A]}, sb f_{\mathbb{I}}(t_E, t)_{[B]} \} \quad (5.14)$$

5.4.3 Interface Generation

In this section we use the $sb f$, together with the DBF of component $\mathbb{C}$, to generate interface $\mathbb{I}$. For component $\mathbb{C}$ to be schedulable using interface $\mathbb{I}$, it is sufficient to ensure that $db f_{MM}(\mathbb{C}, t_I, t_E, t) \leq sb f_{\mathbb{I}}(t_E, t)$ for various time interval lengths. Below we first present the schedulability test for the case when component $\mathbb{C}$ does not experience EMS. That is, the interface only executes in LC mode supplying $C(L)$ resource capacity periodically.

Theorem 7 *A mixed-criticality component $\mathbb{C}$ is schedulable in LC mode with $sb f_{\mathbb{I}}(t_E = t, t)$ if, $\forall t : 0 \leq t \leq t^{max}, \forall t_I : 0 \leq t_I \leq t$,*

$$dbf_{MM}(\mathbb{C}, t_I, t_E, t) \leq sbf_{\mathbb{I}}(t_E = t, t) \text{ If } t_E = t \quad (5.15)$$

where $t^{max} = lcm + \max_{\tau_i \in \mathbb{C}} D_i$ [63], lcm denotes the least common multiple of periods of tasks in $\mathbb{C}$, and $sbf_{\mathbb{I}}(t_E, t)$ is given by Equation (5.11) in Section 5.4.2.

For a given t and t_I , $dbf_{MM}(\mathbb{C}, t_I, t_E, t)$ can be computed using techniques described in Section 5.3.3. The only unknown quantity in Equation (5.15) is the LC resource capacity $C(L)$. To calculate $C(L)$, we need to exploit the fact that there are only two possible values for the floor function in $sbf_{\mathbb{I}}(t_E = t, t)$. Given t , the solution of $sbf_{\mathbb{I}}(t_E = t, t) = dbf_{MM}(\mathbb{C}, t_I, t_E, t)$ denotes the minimal required capacity $C(L)$ so that the execution demand is equal to the resource supply. Let $\Delta = T - ((t - T) \bmod T)$, the value $n = \left\lfloor \frac{t - T + C(L)}{T} \right\rfloor_0$ only has two possible values if (1) $C(L) \geq \Delta$ or (2) $C(L) < \Delta$. Similarly, $[t - 2(T - C(L)) - nT]_0$ can only be equal to 0 if (3) $t - 2(T - C(L)) - nT \leq 0 \Rightarrow C(L) \leq (nT - t + 2T)/2$ or (4) $t - 2(T - C(L)) - nT > 0 \Rightarrow C(L) > (nT - t + 2T)/2$ otherwise. Thus, $sbf_{\mathbb{I}}(t_E = t, t) = dbf_{MM}(\mathbb{C}, t_I, t_E, t)$ can be represented as several simple linear equations. Once we solve the equations, we can choose the value of $C(L)$ among all the solutions which satisfies the predefined conditions. Since we derive different values of $C(L)$ for each t , we need to choose the maximum one among them to ensure the schedulability.

To compute the HC resource capacity $C(H)$, we need to consider the schedulability test when component $\mathbb{C}$ experiences EMS at some time instant $t_E (< t)$. The following theorem presents this test.

Theorem 8 *A mixed-criticality component $\mathbb{C}$ is schedulable in HC mode with $sbf_{\mathbb{I}}(t_E, t)$ if $\forall t : 0 \leq t \leq t^{max}, \forall t_E : 0 \leq t_E \leq t, \forall t_I : 0 \leq t_I \leq t_E$,*

$$dbf_{MM}(\mathbb{C}, t_I, t_E, t) \leq sbf_{\mathbb{I}}(t_E, t) \quad (5.16)$$

where, $t^{max} = lcm + \max_{\tau_i \in \mathbb{C}} D_i$ [63], lcm denotes the least common multiple of periods of tasks in $\mathbb{C}$, and $sbf_{\mathbb{I}}(t_E, t)$ is given by Equation (5.14) in Section 5.4.2.

The only unknown quantity in Equation (5.16) is the HC resource capacity $C(H)$, assuming we have already computed $C(L)$ using Theorem 7. The procedure to calculate $C(H)$ is similar to the procedure to calculate $C(L)$. Equation $sbf_{\mathbb{I}}(t_E, t) = dbf_{MM}(\mathbb{C}, t_I, t_E, t)$ can be represented as some linear equations with the only unknown $C_i(H)$. Once we get the solutions of these equations, we could choose the one that satisfies the conditions.

5.5 Evaluation

In this section we evaluate the performance of the proposed component MC model in terms of offline schedulability as well as its ability to support LC task executions online. Task sets used in the simulation are generated using the following settings, where each parameter is randomly drawn from the given range using an uniform distribution.

- $u_i(L) \in [0.02, 0.1]$.
- $RC = \frac{C_i(H)}{C_i(L)} \in [2, 3]$.
- $T_i \in [10, 150]$.
- $D_i = T_i$ as service adaption strategy, one of the mechanisms being compared, is designed for implicit deadline task systems.
- The probability that task τ_i is a HC task is equal to 0.5.
- For a HC task τ_i , $D_i(L)$ can be tuned by the algorithm in [23].
- For the proposed mechanism, we assume a two component structure, i.e., all the $|H|$ HC tasks are allocated to a HC component $\mathbb{C}_H = \{\mathcal{W}_H, TL_H\}$, and all the LC tasks are allocated to a LC component $\mathbb{C}_L$.

We have chosen relatively small values for $u_i(L)$ and RC so the number of HC tasks $|H|$ in a generated task set would not be too small. This enables us to evaluate the online performance of various approaches when different number of HC tasks synchronously switch to HC mode. The generated task set is evaluated for off-line schedulability as well as on-line performance in terms of support for LC execution under four different mechanisms. These include the mechanism presented in this paper (“Proposed Mechanism”), service adaptation strategy [19] (“Service Adaptation”), Interference Constraint Graph [46] (“ICG”), and the classical mixed-criticality studies in which all the LC jobs are not considered for scheduling once any HC job executes beyond $C_i(L)$ (“Vestal’s model”). In Section 5.5.1 we show the acceptance ratios of different mechanisms, and in Section 5.5.2, we evaluate their capacity to support LC executions through simulations.

5.5.1 Offline Schedulability

In order to generate feasible tasksets, we consider different utilization bounds for the term $U_\tau = \max\{U(L), U(H)\}$. For each bound value, we generate 1000 tasksets based on the procedure described above, and compute their acceptance ratios.

In the elastic model [52], LC tasks have a maximum period T_i^{max} . At runtime, the periods of LC tasks are extended to T_i^{max} even when no HC job exhibits HC behavior. Thus, all LC tasks are only guaranteed to receive $C_i(L)$ time units of budget every T_i^{max} time units. Each LC task has a set of early-release points which enable them to release more frequently. Since the elastic model reduces the dispatch frequency of LC tasks even when no HC task exhibits HC behavior, it is meaningless to evaluate its schedulability performance. The schedulability test for service adaption strategy [31] is based on pessimistic approximate demand bound functions. ICG applies the well known Audsley's algorithm to assign task priorities. Its performance in terms of schedulability is maximized when the interference graph is fully connected, i.e., each HC task has an execution dependency with all LC tasks in the system. For our mechanism, if a hierarchical scheduling framework is considered, then we assume that the MCPR interface period T for both $\mathbb{C}_H$ and $\mathbb{C}_L$ is equal to 5 time units. This is reasonable because the smallest task period in any taskset is 10 time units.

Figures 5.4 and 5.5 show the acceptance ratios of different mechanisms. In Figure 5.4, we show the results of our mechanism for flat scheduling framework. In Figure 5.5, we present results of our mechanism for hierarchical scheduling framework. In these figures, the x-axis denotes the utilization bound value, and the y-axis denotes acceptance ratio, i.e., percentage of task sets deemed schedulable by the different mechanisms. For our mechanism, we consider different values of the tolerance limit $TL_H \in \{0, \lfloor 0.2|H| \rfloor, \lfloor 0.4|H| \rfloor, \lfloor 0.6|H| \rfloor, \lfloor 0.8|H| \rfloor, |H|\}$.

As shown in Figure 5.4, the acceptance ratio of our mechanism clearly depends on the tolerance limit; a higher limit generally implies lower schedulability, because it needs additional resources to support LC executions. When $TL_H \in \{0, \lfloor 0.2|H| \rfloor, \lfloor 0.4|H| \rfloor\}$, our mechanism outperforms both service adaptation and ICG in terms of acceptance ratios. Similar trends can also be observed for our mechanism under a hierarchical framework, except that the schedulability drops more rapidly due to the overhead of hierarchical scheduling. The Vestal's model is represented by the curve with $TL_H = 0$ and it has the best schedulability performance at the cost of providing no support for LC executions at all.

Figure 5.4: Schedulability under a Flat Scheduling Framework

5.5.2 Online Support for LC Executions

In this section, we evaluate the performance of our mechanism in terms of its ability to support LC executions. We define the following quantitative parameter and use it to measure this on-line LC execution support performance.

Definition 13 (Percentage of Finished LC Jobs (PFJ)) Let MAX_t denote the maximum possible number of LC jobs that a task set τ can generate during the time interval $[0, t)$. By definition, $MAX_t = \sum_{L_i=LC} \lceil \frac{t}{T_i} \rceil$. Let FIN_t denote the number of LC jobs that successfully finish by their deadlines in the time interval $[0, t)$ using some mechanism. Then, PFJ is equal to $\frac{FIN_t}{MAX_t}$.

Task sets are generated using the procedure described earlier, and the various mechanisms are simulated to measure their on-line performance. The following additional settings and restrictions are used for this purpose.

Figure 5.5: Schedulability under a Hierarchical Scheduling Framework

- $U_\tau \in \{0.8, 0.85, 0.9\}$.
- Tolerance limit TL_H is chosen to be the largest value that still guarantees schedulability of our mechanism under a flat scheduling framework.
- Task sets are simulated for $t = 10000$ time units.
- Each HC job independently switches to HC mode, i.e., executes for more than LC WCET, with a probability of 0.005, 0.02, 0.05, 0.2 or 0.5.
- The system will transition back to LC mode once there is an idle time instant.

We have chosen a relatively large utilization bound U_τ for each task set, because a small utilization bound indicates that there is sufficient spare capacity for all the mechanisms to support LC executions. In Figures 5.6 and 5.7, we show the corresponding results. The x-axis denotes the probability that a HC job independently switches to HC mode, and the y-axis denotes PFJ for each mechanism. Each point

Figure 5.6: PFJ: $U_\tau = 0.8$

in these figures is calculated by taking an average value of PFJ over 1000 task sets. In Figure 5.6, we consider only those task sets that are deemed to be schedulable by all the presented mechanisms. As shown in the figure, our mechanism consistently outperforms all the other mechanisms for different values of mode switch probability, and the performance gap increases with increasing probability values.

One should note that the results in Figure 5.6 may not be truly representative of the performance of our mechanism in terms of its ability to support LC jobs, and this can be explained as follows. To compare our mechanism's ability to support LC executions with the other mechanisms, we have to use task sets that are schedulable by all these mechanisms. In particular, it does not include many task sets that are schedulable by our mechanism, but not by one of the other mechanisms.

In order to evaluate the ability of our mechanism to support LC executions in a more objective way, we compare the proposed mechanism alone with the Vestal's model, with utilization bound $U_\tau \in \{0.8, 0.85, 0.9\}$. The corresponding simulation results are shown in Figure 5.7. In this case, any task set schedulable by the Vestal's model can be used in the simulation. It can be seen that the performance of

Figure 5.7: PFJ: $U_\tau \in \{0.8, 0.85, 0.9\}$

both our mechanism and the Vestal's model drops when compared with the results in Figure 5.6. However, it can also be seen that, our mechanism still dominates the Vestal's model and the corresponding performance gap does not decrease compared with the gap in Figure 5.6.

5.6 Conclusions and Future Work

To overcome the impractical assumption made in Vestal's model that all LC tasks would be penalized equally once a single HC task executes beyond its LC mode budget $C_i(L)$, in this chapter, we proposed a component-based MC model to offer differentiated service to the LC tasks.

In the proposed model, tasks are assumed to be partitioned into different components and each component has a parameter called tolerance limit that models the expected number of simultaneous violation of $C_i(L)$ by HC tasks. The components are fully isolated as long as no component violates its tolerance limit. We consider EDF-VD* scheduling policy and developed schedulability tests for flat as

well as hierarchical scheduling frameworks. We also propose a MC periodic resource (MCPR) model which abstractly represents the resource demand and supply of a MC component under a hierarchical scheduling framework. We evaluated the performance of the proposed model and test in terms of off-line schedulability and on-line support for LC task executions. Results show that the proposed technique outperforms existing approaches.

However, the component model still has its own limitations. For example, the tolerance limit only models the “number of tasks” that violate $C_i(L)$ simultaneously. In particular, it does not consider the extent to which these tasks exceed $C_i(L)$. Different approaches could be considered if τ_i executes for $C_i(L) + 0.01(C_i(H) - C_i(L))$ time units as opposed to the case when τ_i executes for $C_i(H)$ time units. Besides, once a single HC task exceeds $C_i(L)$, LC tasks in the same component are still dropped immediately. In the next chapter, we address such problems by dynamically allocating execution budgets to the HC tasks.

Chapter 6

Dynamic Budget Management and Reclamation for High-Criticality Tasks

The component-based mixed-criticality (MC) model proposed in Chapter 5 is able to offer differentiated service to the low-criticality (LC) tasks depending on the behavior of specific high-criticality (HC) tasks. However, both in Vestal's as well as this component-based MC models, the LC mode WCET estimations for HC tasks (i.e., $C_i(L)$) are primarily used as static budgets for the HC tasks. They essentially control execution mode-switches, and hence affect the service offered to LC tasks. As discussed in Chapter 1, this needs to specify multiple WCET estimations for a task imposes a significant burden on the application designers. Since these estimations are only used to control execution mode-switches, one may wonder whether it is possible to determine the HC task budgets considering other factors like schedulability as opposed to using WCET estimations. This will not only reduce the burden on the designer, but may also improve schedulability performance and support for LC tasks if the budgets are chosen carefully.

Addressing the above issues, in this chapter, we introduce a *dynamic MC task and scheduling model* that redefines the mode-switch condition using dynamic budget allocations for HC tasks. We focus on dual-criticality systems scheduled on a uniprocessor platform. In the dynamic model, LC mode budgets for individual HC tasks are no longer equal to $C_i(L)$. In fact, each task only has one WCET estimation given by the designer, and this is used to ensure a safe execution for the HC tasks. That is, HC tasks are always guaranteed to receive resources as much as their respective WCET estimations. The LC budgets of HC tasks are then determined at runtime based on a system-wide total LC budget allocation for all HC tasks combined. This total budget value is determined off-line based on schedulability considerations.

To further enhance the service for LC tasks, the dynamic model also allows guaranteed service for these tasks after an execution mode-switch. This model and technique has been previously published in [64].

In Section 6.2.1, we present the new dynamic MC task and scheduling model. In Section 6.2.2, we propose a new scheduling policy called Earliest Deadline First - Universal Virtual Deadlines (EDF-UVD), which is a variant of EDF-VD [31]. In Section 6.3, we propose a runtime technique to distribute budgets to jobs of HC tasks in the LC mode, depending on job execution requirements as well as the fixed total LC budget value. In Section 6.4, we propose a budget reclamation scheme that could work along with the proposed budget allocation strategy to further postpone or prevent the occurrence of mode-switches. The schedulability analysis for the dynamic model is presented Section 6.5. We present some guidelines on how to determine the total LC budget value for HC tasks in Section 6.6. Finally, in Section 6.7, we present the evaluation results for our proposed technique. Before we continue, we list the notations that would be used in this chapter.

Table 6.1: Notations used in Chapter 6

$B_i(L)/B_i(H)$	LC/HC budget of τ_i
τ^L	subset of LC tasks in τ
τ^H	subset of HC tasks in τ
$D_i(L)$	virtual deadline of task τ_i
$u_i = \frac{C_i}{T_i}$	the utilization of task τ_i
$U_L = \sum_{\tau_i \in \tau^L} u_i$	the utilization of τ^L
$U_H = \sum_{\tau_i \in \tau^H} u_i$	the utilization of τ^H
α_i	the proportion of utilization that is statically reserved for LC task τ_i in HC mode
α^*	the proportion of U_L that is reserved for all LC tasks combined in HC mode
β^*	the proportion of U_H that is reserved for all HC tasks combined in LC mode
$J_{i,j}$	the j_{th} job of τ_i
$r_{i,j}$	release time of $J_{i,j}$
$d_{i,j}$	absolute deadline of $J_{i,j}$
$d_{i,j}(L) = r_{i,j} + D_i(L)$	the absolute virtual deadline
$b_{i,j}$	the budget allocated to the j_{th} job of τ_i
$e_{i,j}$	the received cpu time of $J_{i,j}$, which is also budget consumed by $J_{i,j}$
e_i^m	the maximum amount of cpu time for which any job of τ_i has received in the current busy interval
$E_{i,j}$	the execution requirement of $J_{i,j}$
$\phi_{i,j}$	the amount of consumed unused budget (U-budget) by $J_{i,j}$
$o_{i,j}$	the amount of consumed own budget (O-budget) by $J_{i,j}$.
o_i^m	the maximum amount of O-budget that any job of τ_i has consumed in the current busy interval

6.1 Motivation and Contribution

In Vestal’s dual-criticality model, the LC budget of a HC task τ_i ($B_i(L)$) is equal to its WCET estimation for the LC mode $C_i(L)$ provided by the designer. Since Vestal’s model uses a static (fixed) budget allocation in the LC mode; $B_i(L) = C_i(L)$, we can denote this model as the **static model**. As discussed earlier, existing studies have not considered other techniques for determining these LC budgets. For example, is it possible that with fine-grained management of these budgets, the execution mode-switch can be postponed or even completely avoided? Can these budgets be determined at runtime depending on job behaviors in the recent past? Can these budgets be determined based on schedulability considerations?

Answering the above questions, in this chapter, we propose a more dynamic MC task and scheduling model based on the following principle: *HC tasks can be allocated budgets in the LC mode dynamically at runtime depending on the overall processing requirements of the MC system. As long as allocated budgets continue to ensure schedulability for all the tasks, it should be safe to do so.* In short, we denote this model as the **dynamic model**.

In the dynamic model, the application designers are not required to specify $C_i(L)$ any more, thus reducing their burden. Off-line, we determine a **total LC budget** allocation (single value) for all HC tasks combined. At runtime, we use a dynamic budget allocation strategy to distribute this total LC budget to individual jobs of HC tasks. A system-wide budget for HC tasks in the LC mode, as opposed to budgets for individual tasks as in the static model, allows more flexibility at runtime to distribute this budget depending on job execution requirements. In the dynamic model, a mode-switch will only be triggered if the total LC budget is exceeded. As a result, the dynamic model redefines the mode-switch condition and the mode-switch from LC to HC can also be potentially postponed.

To further delay mode-switches, based on the observation that many jobs may complete prior to their allocated LC budget, we also propose a budget reclamation scheme which could work along with the dynamic budget allocation strategy. With the budget reclamation scheme, jobs at runtime would first consume the unused LC budget from jobs with higher priority. As a result, HC jobs are less likely to consume their own LC budget, and the mode-switch will be less likely to be triggered. We show that the proposed budget allocation strategy is optimal in terms of its ability to postpone the mode-switch when compared to any static scheme. We also derive a schedulability test for the proposed model under EDF-UVD scheduling policy, and show through experiments that the dynamic model outperforms the static model in terms of its ability to support the execution of LC tasks.

6.2 Dynamic MC Task and Scheduling Model

6.2.1 Task Model

In the dynamic MC task model, a LC task is defined as a tuple $(T_i, C_i, L_i = LC, \alpha_i)$ and a HC task is defined as a tuple $(T_i, C_i, L_i = HC)$, where T_i is the minimum separation time between two consecutive job releases, L_i denotes the criticality level, and C_i upper bounds the WCET of the task. Thus the application designer only needs to provide one WCET estimation C_i as in the standard non-MC task model. Thus, in the dynamic model, C_i is equivalent to $C_i(H)$ (if τ_i is a HC task) or $C_i(L)$ (if τ_i is a LC task).

For a LC task τ_i , its additional parameter $\alpha_i \in [0, 1]$ denotes the proportion of utilization that is statically reserved for τ_i in HC mode. Thus, its HC budget is also fixed at $B_i(H) = C_i \times \alpha_i$ ($\alpha_i \in [0, 1]$), while its LC budget is fixed at $B_i(L) = C_i$. For a HC task τ_i , its HC budget is fixed at $B_i(H) = C_i$ similar to the static model, but the LC budget of jobs released by τ_i is determined dynamically at runtime. It varies depending on the past execution demand of all HC jobs. We focus on implicit deadline task systems (i.e., $D_i = T_i$ for each task), and consider the problem of scheduling n such tasks $\tau = \{\tau_1, \dots, \tau_n\}$ on a uniprocessor platform.

Similar to the static model, we can also define a schedulability criteria for algorithms that schedule task systems based on the dynamic model presented above. Since LC tasks also receive guaranteed budgets in HC mode, unlike the static model, their deadlines cannot be ignored. Further, since individual HC tasks do not have a fixed budget in the LC mode, the scheduler only needs to ensure that each job of HC task τ_i receives as much execution as it needs up to C_i time units in both modes. If it is able to provide a budget of C_i for some job of τ_i (but still ensure schedulability for all the tasks) while remaining in the LC mode, then it is free to do so. In fact, this flexibility in budget allocation is a key advantage of the dynamic model.

Definition 14 (MC-Schedulable in Dynamic Model) *A task system τ in the dynamic model is defined to be **MC-Schedulable** by a scheduling algorithm if the following two conditions hold:*

1. *Jobs of LC task τ_i with deadline in the LC mode receive up to $B_i(L)(= C_i)$ units of budget each by their deadlines, and jobs of LC task τ_i with deadline in the HC mode receive up to $B_i(H)(= \alpha_i \times C_i)$ units of budget each by their deadlines.*
2. *Jobs of HC task τ_i receive as much budget as they need by their deadlines as long as it does not exceed C_i .*

6.2.2 EDF-UVD Scheduling Strategy

We propose a new scheduling strategy for the dynamic model called EDF-UVD (Earliest Deadline First with Universal Virtual Deadlines). EDF-UVD is a variant of the well known algorithm EDF-VD [31], and similarly uses a single factor to set virtual deadlines. The only difference between EDF-VD and EDF-UVD is that under EDF-UVD, even LC jobs would be assigned virtual deadlines shorter than their original deadlines in the LC mode. A job of LC task τ_i would be scheduled based on its virtual deadline as long as it has not executed for more than $\alpha_i \times C_i$ time units which is the amount of execution demand that must be met even in the HC mode. Thereafter, it will be scheduled using its actual deadline. Thus, EDF-UVD can be formally defined as follows.

1. Let $D_i(L) = x \times T_i$, where $x \in [0, 1]$, denote the virtual deadline for each task $\tau_i \in \tau$.
2. When the system is in the LC mode:
 - A job of HC task τ_i will be scheduled based on its virtual deadline $D_i(L)$.
 - A job of LC task τ_i that has executed for less than $\alpha_i \times C_i$ time units will be scheduled based on its virtual deadline $D_i(L)$.
 - A job of LC task τ_i that has executed for $\alpha_i \times C_i$ time units or more will be scheduled based on its original deadline D_i .
3. When the system is in the HC mode, both LC and HC jobs will be scheduled using their actual deadlines.
4. In each mode, all jobs will be scheduled by EDF scheduling policy.

EDF-UVD can reduce unnecessary budget allocations for LC tasks in comparison to EDF-VD. This happens when some job of a LC task τ_i executes beyond $\alpha_i \times C_i$ time units in the LC mode, and the system switches mode before the deadline of this job. In this case, any execution of the job beyond $\alpha_i \times C_i$ is unnecessary, because it is not required to be satisfied based on the definition of MC-schedulable. However, it is impossible to know prior to the mode-switch which execution is unnecessary. To address this issue, we use virtual deadlines for LC jobs in the LC mode. The main intuition behind this strategy is that the first $\alpha_i \times C_i$ execution units would be scheduled with a higher priority (based on virtual deadline), and the remaining execution units would be scheduled with a normal priority (based on original deadline). Thus, all jobs of LC tasks would prioritize the first $\alpha_i \times C_i$ execution units, thereby completing those execution units earlier than in the case of EDF-VD. Below we use a simple example to illustrate how EDF-UVD reduces unnecessary budget allocations.

Example 4 As shown in Figure 6.1(a), there are two LC jobs J_1^* and J_2^* that are released before t^* but have deadline after t^* , where t^* denotes the mode-switch instant. If J_1^* receives C_1 units of budget before t^* , then deadline miss would happen because J_2^* receives less than $\alpha_2 \times C_2$ units of budget before its deadline D_2 . However as shown in Figure 6.1(b), after J_1^* executes for $\alpha_1 \times C_1$ units of time, it is scheduled using its true deadline T_1 , and at that time J_2^* would have higher priority than J_1^* . As a result, the $(1 - \alpha_1) \times C_1$ units of budget previously consumed by J_1^* is now consumed by J_2^* , and the deadline miss at D_2 is avoided.

Figure 6.1: Example in which EDF-UVS performs better than EDF-VD

6.3 Maximum Execution-based Budget Allocation (MEBA)

In this section we propose an efficient runtime budget allocation scheme named MEBA (short for Maximum Execution-based Budget Allocation) for distributing the LC budget among jobs of HC tasks. Let τ^H denote subset of high-criticality (HC) tasks and β^* denote the service level of τ^H in LC mode, which is equal to the proportion of U_H that is reserved as total budget for all HC tasks combined in the

LC mode. While distributing the LC budget among jobs of HC tasks, we ensure that the total LC budget allocated to all HC tasks over any period of time is proportional to $\beta^* \times U_H$ ¹.

This strategy also determines when a mode-switch will occur in the dynamic model; it is precisely when the total demand of all HC tasks during the current busy interval exceeds $\beta^* \times U_H$. To be able to do this efficiently, we need to store some information about budgets consumed by HC jobs in the past. We first use a simple example to illustrate MEBA, and then present the details about it.

Figure 6.2: Example illustrating MEBA

Example 5 As shown in Figure 6.2, a system $\tau = \{\tau_1, \tau_2, \tau_3\}$ is scheduled by EDF-UVD and MEBA, where $\tau_1 = (10, 4, HC)$, $\tau_2 = (10, 4, HC)$, $\tau_3 = (10, 6, LC, \alpha_3 = 0)$ and $\beta^* = 0.5$. Before $J_{1,1}$ begins execution at $r_{1,1} = 0$, we allocate a budget to it as follows.

$$\frac{b_{1,1}}{T_1} = \beta^* \times U_H \Rightarrow b_{1,1} = T_1 \times \beta^* \times U_H = 10 * 0.5 * 0.8 = 4.$$

Thus $J_{1,1}$ is allocated budget proportional to $\beta^* \times U_H$. When $J_{1,1}$ completes after executing for $e_{1,1} = 2$ time units, $e_{1,1}$ will be used to determine $b_{2,1}$, the budget for $J_{2,1}$. Thus, before $J_{2,1}$ begins execution at

¹In this and the following section we assume that the parameter β^* is known and present runtime budget allocation strategies for HC tasks. Later, we derive a schedulability condition for the dynamic MC task model, assuming it is scheduled under EDF-UVD and the runtime strategies presented in these sections. This condition can be used to determine a safe value for β^* that ensures the resulting system is MC-Schedulable.

$t = 2$, we allocate a budget to it as follows.

$$\frac{e_{1,1}}{T_1} + \frac{b_{2,1}}{T_2} = \beta^* \times U_H \Rightarrow b_{2,1} = T_2 \times \left(\beta^* \times U_H - \frac{e_{1,1}}{T_1} \right) = 2$$

A fraction of budget equal to $e_{1,1}/T_1 = 0.2$ is reserved for future jobs of task τ_1 . The remaining budget of $(\beta^* \times U_H - e_{1,1}/T_1)$ is allocated to $J_{2,1}$.

When $J_{2,1}$ completes after executing for $e_{2,1} = 1$ time unit, $e_{2,1}$ will be used to determine $b_{1,2}$, the budget for $J_{1,2}$. As a result, at $r_{1,2} = 10$, a new value for $b_{1,2}$, which is now the budget for job $J_{1,2}$, will be computed as follows.

$$\frac{b_{1,2}}{T_1} + \frac{e_{2,1}}{T_2} = \beta^* \times U_H \Rightarrow b_{1,2} = T_1 \times \left(\beta^* \times U_H - \frac{e_{2,1}}{T_2} \right) = 3$$

Similarly, when $J_{1,2}$ completes after executing for $e_{1,2} = 1$ time unit, $b_{2,2}$ will again be updated depending on $\max\{e_{1,1}, e_{1,2}\}$. To ensure that the total allocation is proportional to $\beta^* \times U_H$, $b_{2,2}$ is updated as follows.

$$b_{2,2} = T_2 \times \left(\beta^* \times U_H - \frac{\max\{e_{1,1}, e_{1,2}\}}{T_1} \right) = 2$$

Since $J_{2,2}$ has an execution requirement as much as $E_{2,2} = 3(> b_{2,2})$ time units, a mode-switch to HC mode is triggered at $t = 13$.

In the above example, we reserve $e_1^m = \max\{e_{1,1}, e_{1,2}\}$ units of budget for future jobs of τ_1 when determining the budget for the second job of τ_2 , i.e.,

$$\frac{b_{2,2}}{T_2} = \left(\beta^* \times U_H - \frac{\overbrace{e_1^m = \max\{e_{1,1}, e_{1,2}\}}^{\text{budget reserved for future jobs of } \tau_1}}{T_1} \right)$$

Thus, at all times we need to store for each HC task τ_i the **maximum amount of cup time** for which any job of τ_i has received in the **current busy interval**, i.e., $e_i^m = \max\{e_{i,1}, e_{i,2}, \dots\}$. We reserve a

budget proportional to this largest received budget for all future jobs of that task.

For each HC task τ_i , initialize $e_i^m = 0$. Let $b_{i,j}$ denote the budget allocated to the j_{th} job of τ_i . The MEBA budget allocation strategy comprises four functions as shown in Algorithm 4

- Function COMPUTE_BI computes the LC budget $b_{i,j}$ for $J_{i,j}$.
- Function TRIGGER is used to switch the system mode from LC to HC.
- Function UPDATE_E is used to record the value of e_i^m .
- Function RESET_E is used to reset the value of e_i^m .

Now we present the situations in which each function in MEBA should be executed. RESET_E would be executed if the system experiences an idle instant. Figure 6.3 shows the situations in which Function COMPUTE_BI, TRIGGER and UPDATE_E should be executed. Suppose HC job $J_{i,j}$ is dispatched to the processor by the scheduler, then function COMPUTE_BI will be executed before $J_{i,j}$ starts to execute. Function TRIGGER will be executed if HC job $J_{i,j}$ has consumed $b_{i,j}$ units of LC budget in total but does not complete. Function UPDATE_E would be executed when HC job $J_{i,j}$ gets preempted or completes at the current time instant.

Figure 6.3: Situations to call functions in MEBA

The following lemma records an important property of MEBA.

Lemma 7 (Mode-switch condition) *Suppose a mode-switch to HC mode is triggered at some time instant t^* in a **busy interval** under MEBA. For each HC task τ_i and each time instant $t \leq t^*$, let $e_{i,t}^m$ denote the maximum amount of time for which any job of τ_i has executed before t in that busy interval. Then, during this busy interval, the system has the following two properties:*

Algorithm 4 MEBA

- 1: **function** COMPUTE_BI $\triangleright$ This function is called when HC job $J_{i,j}$ is allocated to the processor at the current time instant.
- 2: **if** System Mode=LC **then**
- 3: Before $J_{i,j}$ starts to execute, update LC budget

$$b_{i,j} = T_i \times \left(\beta^* \times U_H - \sum_{\tau_k \in \tau^H \setminus \tau_i} \frac{e_k^m}{T_k} \right)$$

- 4: **end if**
- 5: **end function**
- 6:

- 7: **function** TRIGGER $\triangleright$ This function is called when HC job $J_{i,j}$ has consumed $b_{i,j}$ units of LC budget in total but does not complete.
- 8: Switch System Mode to HC.
- 9: **end function**
- 10:

- 11: **function** UPDATE_E $\triangleright$ This function is called if HC job $J_{i,j}$ gets preempted or completes at the current time instant.
- 12: Update

$$e_i^m = \max \{e_i^m, e_{i,j}\}$$

where $e_{i,j}$ denote the total budget consumed by $J_{i,j}$ so far.

- 13: **end function**
 - 14:

 - 15: **function** RESET_E $\triangleright$ This function is called if the system experiences an idle instant.
 - 16: Reset $e_i^m = 0$ for each HC task τ_i .
 - 17: **end function**
-

Property 1: For all $t \leq t^*$,

$$\sum_{\tau_i \in \tau^H} \frac{e_{i,t}^m}{T_i} \leq \beta^* \times U_H. \quad (6.1)$$

Property 2: Further, t^* is the earliest time instant in that busy interval such that there is a job J_{k,h^*} of some HC task τ_k with execution requirement $E_{k,h^*} > e_{k,t^*}^m$, and

$$\sum_{\tau_i \in \tau^H} \frac{e_{i,t^*}^m}{T_i} = \beta^* \times U_H. \quad (6.2)$$

Proof: **Proof for Property 1:** Since a busy interval only has one mode-switch to HC mode under MEBA², t^* is the only such mode-switch instant in the busy interval under consideration.

²After a system switches to HC mode, it would revert to LC mode only after an idle instant.

We will prove property 1 by contradiction: Suppose Equation (6.1) does not hold, and $t_f(\leq t^*)$ denotes the earliest time instant where it fails. By proving that a mode-switch should have already occurred before t_f , we can create a contradiction because t^* is assumed to be the only mode-switch in the busy interval under consideration.

Let J_{k,h^*} denote a job of HC task τ_k that was continuously scheduled in the interval $[t', t_f)$ for some $t' < t_f$. The value of b_{k,h^*} was updated at t' to $T_k \times (\beta^* \times U_H - \sum_{\tau_i \in \tau^H \setminus \tau_k} e_{i,t'}^m / T_i)$ and remains the same up to t_f . Further, since t_f is the earliest time instant at which Equation (6.1) fails, we know that $e_{k,t'}^m \leq T_k \times (\beta^* \times U_H - \sum_{\tau_i \in \tau^H \setminus \tau_k} e_{i,t'}^m / T_i) = b_{k,h^*}$. Further, $\forall \tau_i \in \tau^H \setminus \tau_k, e_{i,t_f}^m = e_{i,t'}^m$, because none of these tasks executed in the interval $[t', t_f)$. Then, since Equation (6.1) fails at t_f , we get,

$$\begin{aligned} e_{k,t_f}^m &> T_k \left(\beta^* \times U_H - \sum_{\tau_i \in \tau^H \setminus \tau_k} e_{i,t_f}^m / T_i \right) \iff \\ e_{k,t_f}^m &> T_k \left(\beta^* \times U_H - \sum_{\tau_i \in \tau^H \setminus \tau_k} e_{i,t'}^m / T_i \right) \iff e_{k,t_f}^m > b_{k,h^*} \end{aligned}$$

Since $e_{k,t'}^m \leq b_{k,h^*} < e_{k,t_f}^m$ and J_{k,h^*} is the only job executing in $[t', t_f)$, it must be the case that J_{k,h^*} itself has executed for a total time of e_{k,t_f}^m by t_f . This, combined with the fact that J_{k,h^*} did not execute for more than $e_{k,t'}^m$ before t' , indicates that there is some time instant in $[t', t_f)$ when J_{k,h^*} has completed b_{k,h^*} units of execution but remains incomplete. By definition of MEBA, this would have triggered a mode-switch to HC mode at that time instant. This is impossible however, because $t^*(\geq t_f)$ is the only mode-switch in the busy interval. Thus we have shown that Equation (6.1) holds for all $t \leq t^*$.

Proof for Property 2: Since a mode-switch is triggered at t^* , from the definition of MEBA we know that there is a job J_{k,h^*} of some HC task τ_k that remains incomplete at t^* . By t^* , J_{k,h^*} has executed for exactly $\lambda_{k,t^*} = b_{k,h^*} = T_k \times (\beta^* \times U_H - \sum_{\tau_i \in \tau^H \setminus \tau_k} e_{i,t^*}^m / T_i)$ time units. Therefore at t^* , we have $\sum_{\tau_i \in \tau^H \setminus \tau_k} e_{i,t^*}^m / T_i + \lambda_{k,t^*} / T_k = \beta^* \times U_H$, and the total execution demand E_{k,h^*} of J_{k,h^*} is greater than λ_{k,t^*} . Then $e_{k,t^*}^m = \lambda_{k,t^*}$, because Equation (6.1) holds at t^* and $e_{k,t^*}^m \geq \lambda_{k,t^*}$ by definition. Thus we have identified the conditions that must hold when a mode-switch is triggered under MEBA. Since t^* is the only mode-switch in this busy interval, we can conclude that t^* is the earliest time instant when these conditions are satisfied. ■

Finally, the following lemma shows that for a given β^* , MEBA is optimal in terms of its ability

to postpone the mode-switch when compared to any other offline or runtime strategy that uses a fixed budget for each HC task in the LC mode.

Lemma 8 *For the dynamic MC task system τ , consider any budget allocation $B_i(L)(\forall \tau_i \in \tau^H)$ satisfying the condition $\sum_{\tau_i \in \tau^H} B_i(L)/T_i \leq \beta^* \times U_H$, such that each job of each HC task τ_i is given a budget of exactly $B_i(L)$ in the LC mode. Then, for any job sequence (release time and execution demand) of τ , the mode-switch instant based on this budget allocation is no later than the mode-switch instant under MEBA.*

Proof: We prove this lemma by contradiction. Suppose under MEBA the mode-switch happens at t^* for some job sequence. From Lemma 7 we know that

$$\sum_{\tau_i \in \tau^H} e_{i,t^*}^m / T_i = \beta^* \times U_H,$$

where e_{i,t^*}^m denotes the maximum amount of budget for which any job of τ_i has received before t^* in the present busy interval. Also, there is at least one HC task τ_k having an incomplete job J_{k,h^*} at t^* with execution demand $E_{k,h^*} > e_{k,t^*}^m$.

Suppose there exists a budget allocation satisfying the condition

$$\sum_{\tau_i \in \tau^H} B_i(L)/T_i \leq \beta^* \times U_H$$

that can further postpone the mode-switch. Then it must be the case that $\forall \tau_i \in \tau^H : B_i(L) \geq e_{i,t^*}^m$ and $B_k(L) > e_{k,t^*}^m$. However this is impossible because the assumption $\sum_{\tau_i \in \tau^H} B_i(L)/T_i \leq \beta^* \times U_H$ is violated in that case. ■

For a given MC task system, the optimality in Lemma 8 is valid in comparison to all existing methods in MC studies that use the static model (e.g., [14, 23, 24, 29–31, 59]), when the fixed LC budgets of HC tasks satisfy the condition $\sum_{\tau_i \in \tau^H} B_i(L)/T_i \leq \beta^* \times U_H$. So, irrespective of the specific LC budget values, as long as the total allocated budget satisfies this condition, the mode-switch under our proposed approach will be no earlier than under the existing approach.

6.4 MEBA with Budget Reclamation (MEBA+)

An important advantage of the dynamic model is that it can reduce the probability that a system switches to HC mode when compared to the static model. However, the mode-switch condition in Equation 6.1, i.e.,

$$\sum_{\tau_i \in \tau^H} \frac{e_i^m}{T_i} \leq \beta^* \times U_H$$

is more likely to be violated when the system experiences a long busy interval because the value of e_i^m can only be reset to 0 after an idle instant. Therefore, in this section, we propose a budget reclamation scheme to further prevent the occurrence of mode-switch.

Previous works e.g. [65] usually exploit the unused budget of hard real-time tasks to serve soft real-time tasks, and soft real-time tasks would run under the control of a pseudo hard real-time task. These works aim at utilizing the unused budget to serve soft real-time tasks in addition to the hard real-time tasks. In contrast, the budget reclamation scheme proposed in this section aims at preventing the occurrence of mode-switch.

Suppose a job $J_{i,j}$ has consumed $e_{i,j}$ units of LC budget (i.e., $J_{i,j}$ has received $e_{i,j}$ units of budget), then the $e_{i,j}$ units of budget can be classified into two categories:

1. **Definition 15 (Unused Budget (U-budget))** *Among the $e_{i,j}$ units of budget consumed by $J_{i,j}$, there are $\phi_{i,j}$ units of unused budget from jobs with higher priority. The $\phi_{i,j}$ units of U-budget are initially allocated to other jobs with higher priorities than $J_{i,j}$. However, these jobs do not consume them, and later $J_{i,j}$ consumed these U-budget.*
2. **Definition 16 (Own Budget (O-budget))** *Among the $e_{i,j}$ units of budget consumed by $J_{i,j}$, there are $o_{i,j}$ units of its own budget, i.e., budget allocated $J_{i,j}$ itself, where $o_{i,j} = e_{i,j} - \phi_{i,j}$.*

The technique to calculate the value of $o_{i,j}$ and $\phi_{i,j}$ will be presented in Section 6.4.1, and here we assume these values are known.

With the budget reclamation scheme, (i.e. in MEBA+), we will use the O-budget ($o_{i,j}$) instead of total budget ($e_{i,j}$) to control mode-switch. That is, a HC job will trigger a mode-switch only when it has consumed $b_{i,j}$ units of its O-budget (i.e., $o_{i,j} = b_{i,j}$), but does not signal completion. Hence, the function used to control mode-switch in MEBA+, TRIGGER+, is modified as shown in Algorithm 5.

Let o_i^m denote the maximum amount of O-budget that any job of τ_i has consumed in the **latest busy interval**. Then function COMPUTE_BI+ in MEBA+ would use o_i^m instead of e_i^m to determine $b_{i,j}$ for

Algorithm 5 TRIGGER+

- 1: **function** TRIGGER+ ▷ This function is called when HC job $J_{i,j}$ has consumed $b_{i,j}$ units of O-budget without completion.
 - 2: Switch System Mode to HC.
 - 3: **end function**
-

$J_{i,j}$. The details of COMPUTE_BI+ is shown in Algorithm 6.

Algorithm 6 COMPUTE_BI+

- 1: **function** COMPUTE_BI+
- 2: **if** System Mode=LC **then**
- 3: Before $J_{i,j}$ starts to execute at the current time instant, update LC budget

$$b_{i,j} = T_i \times \left(\beta^* \times U_H - \sum_{\tau_k \in \tau^H \setminus \tau_i} \frac{o_k^m}{T_k} \right) \quad (6.3)$$

- 4: **end if**
 - 5: **end function**
-

In MEBA+, function UPDATE_E+ will update the value of $o_i^m = \max\{o_i^m, o_{i,j}\}$ if HC job $J_{i,j}$ gets preempted or completes at the current time instant. Similarly, function RESET_E+ in MEBA+ will reset the value of o_i^m to 0 once an idle instant occurs. The details of UPDATE_E+ and RESET_E+ are shown in Algorithm 7.

Algorithm 7 UPDATE_E+ and RESET_E+

- 1: **function** UPDATE_E+ ▷ This function is called if HC job $J_{i,j}$ gets preempted or completes at the current time instant.
 - 2: Update

$$o_i^m = \max\{o_i^m, o_{i,j}\}$$
 - 3: _____
 - 4: **end function**
 - 5: **function** RESET_E+ ▷ This function is called if the system experiences an idle instant.
 - 6: Reset $o_i^m = 0$ for each HC task τ_i .
 - 7: **end function**
-

We have already introduced the way that MEBA+ determines the LC budget ($b_{i,j}$) of HC jobs and controls the mode-switch according to the O-budget ($o_{i,j}$) consumed by each job. In the following section, we present techniques to calculate the value of $o_{i,j}$.

6.4.1 Functions to Refill and Reclaim U-budget

In this section, we present the techniques to calculate the value of $o_{i,j}$, by keeping track of the unused budgets (U-budget) of completed jobs. We first define a budget queue (BQ) which is used to store the U-budget of completed jobs. Note that, U-budget can be generated by both LC as well as HC jobs when they finish before their allocated budget in the LC mode.

Definition 17 (BQ) *Budget queue (BQ) is a queue of tuples $\chi = (g, d)$, where g denotes the amount of the U-budget, and d is the deadline which is also the expiration time of this budget. BQ is sorted by the deadline of tuples $(\chi.d)$ in ascending order.*

6.4.1.1 Budget Refill

Function REFILL is shown in Algorithm 8, and it would be executed only if some job $J_{i,j}$ completes at the current time instant. Let $d_{i,j}(L) = r_{i,j} + D_i(L)$ denote the absolute virtual deadline and $d_{i,j} = r_{i,j} + T_i$ denote the absolute actual deadline for job $J_{i,j}$.

For a LC job, its LC budget $B_i(L)$ is fixed, but its LC budget has two priorities under EDF-UVD. First, if $o_{i,j} < \alpha_i \times B_i(L)$, then $\alpha_i \times B_i(L) - o_{i,j}$ units of budget has virtual deadline $d_{i,j}(L)$ while the remaining $(1 - \alpha_i) \times B_i(L)$ units of budget has deadline $d_{i,j}$. As shown in line 4-6 of Algorithm 8, $(\alpha_i \times B_i(L) - o_{i,j}, d_{i,j}(L))$ and $((1 - \alpha_i) \times B_i(L), d_{i,j})$ will be inserted into BQ in this case.

Otherwise if $o_{i,j} > \alpha_i \times B_i(L)$, then the $B_i(L) - o_{i,j}$ units of budget only has deadline $d_{i,j}$, and hence $(B_i(L) - o_{i,j}, d_{i,j})$ will be inserted into BQ (line 7-9 of Algorithm 8).

On the other hand, a HC job does not have a fixed LC budget. Suppose HC job $J_{i,j}$ completes at the current time instant. Since at least o_i^m units of LC budget is reserved for future jobs of τ_i , a tuple $(o_i^m - o_{i,j}, d_{i,j}(L))$ is inserted into BQ if $o_{i,j} < o_i^m$ (line 11-13 of Algorithm 8).

6.4.1.2 Budget Reclaim

The underlying tenet of the budget reclamation step is that the budget in BQ from jobs with highest priority (earliest deadline) will always be consumed first. However, we should note that the U-budget of $\chi \in BQ$ would be expired by its deadline $\chi.d$.

Before job $J_{i,j}$ starts to execute, $o_{i,j}$ would be initialized to 0. As $J_{i,j}$ executes from t_s to t_e **continuously**, $o_{i,j}$ would also increase correspondingly (i.e., $o_{i,j} = o_{i,j} + (t_e - t_s)$). Suppose function

Algorithm 8 REFILL($J_{i,j}, BQ$)

```

1: function REFILL( $J_{i,j}, BQ$ )  ▷ This function is called if  $J_{i,j}$  completes at the current time instant.
2:   if  $J_{i,j}$  completes at the current time instant then
3:     if  $L_i = LC$  and  $o_{i,j} < B_i(L)$  then
4:       if  $o_{i,j} < B_i(L) \times \alpha_i$  then
5:         Insert  $(\alpha_i \times B_i(L) - o_{i,j}, d_{i,j}(L))$  into BQ.
6:         Insert  $((1 - \alpha_i) \times B_i(L), d_{i,j})$  into BQ.
7:       else
8:         Insert  $(B_i(L) - o_{i,j}, d_{i,j})$  into BQ.
9:       end if
10:    end if
11:    if  $L_i = HC$  and  $o_{i,j} < o_i^m$  then
12:      Insert  $(o_i^m - o_{i,j}, d_{i,j}(L))$  into BQ.
13:    end if
14:  end if
15: end function

```

RECLAIM in Algorithm 9 is called at t_e and it has identified that there are $\phi_{i,j}$ units of U-budget among the $t_e - t_s$ units of budget that $J_{i,j}$ has consumed. Then $o_{i,j}$ is updated as follows,

$$o_{i,j} = o_{i,j} - \phi_{i,j} \quad (6.4)$$

Now we introduce how to calculate the amount of U-budget ($\phi_{i,j}$) among the $t_e - t_s$ units of total budget consumed by $J_{i,j}$ during $[t_s, t_e]$. In RECLAIM, $\phi_{i,j}$ is initialized to 0, and then we loop through BQ to search for an eligible tuple $\chi \in BQ$ with earlier deadline than $d_{i,j}$.

1. If $\chi.g + \phi_{i,j} < t_e - t_s$, then all the U-budget of χ would be consumed by $J_{i,j}$. In this case, we update

$$\phi_{i,j} = \phi_{i,j} + \chi.g$$

and remove χ from BQ (line 10-12 of Algorithm 9). Then we go to the next iteration.

2. Else if $\chi.g + \phi_{i,j} \geq t_e - t_s$, then at most $t_e - t_s - \phi_{i,j}$ units of U-budget of χ can be consumed by $J_{i,j}$ by t_e . Thus, we update $\chi.g = \chi.g - (t_e - t_s - \phi_{i,j})$ and $\phi_{i,j} = t_e - t_s$ (line 13-16 of Algorithm 9). After this, we would jump out of the loop, and return $o_{i,j} - \phi_{i,j}$ which is the amount of O-budget consumed by $J_{i,j}$ by t_e .

Algorithm 9 RECLAIM

```

1: function RECLAIM( $J_{i,j}, BQ, t_s, t_e$ )  $\triangleright t_s$  and  $t_e$  are two global variables.
2:   for  $\chi \in BQ$  do
3:     if  $\chi.d \leq t_s$  then
4:       remove  $\chi$  from BQ
5:     end if
6:   end for
7:    $\phi_{i,j} = 0$ 
8:   for  $\chi \in BQ$  do
9:     if  $d_{i,j} \geq \chi.d$  then
10:      if  $\chi.g + \phi_{i,j} < t_e - t_s$  then
11:         $\phi_{i,j} = \phi_{i,j} + \chi.g$ 
12:        remove  $\chi$  from BQ
13:      else
14:         $\chi.g = \chi.g - (t_e - t_s - \phi_{i,j})$ 
15:         $\phi_{i,j} = t_e - t_s$ 
16:        break
17:      end if
18:    else
19:      break
20:    end if
21:  end for
22:  return  $o_{i,j} - \phi_{i,j}$ 
23: end function

```

6.4.2 Execution Flows of Functions in MEBA+

In this Section, we present the situations in which functions in MEBA+ should be executed. RE-SET-E is executed only if the system experiences an idle instant. In Figure 6.4, we show the execution flow of functions COMPUTE-BI+, TRIGGER+, UPDATE-E+, REFILL and RECLAIM.

Once a new job $J_{i,j}$ is released, it is waiting for execution in the ready queue. Suppose $J_{i,j}$ is dispatched to the processor by the scheduler and $L_i = HC$, then COMPUTE-BI+ is executed before it starts to execute. Suppose $J_{i,j}$ starts to execute at t_s and completes at t_e , then RECLAIM($J_{i,j}, BQ, t_s, t_e$), UPDATE-E+ (only if $J_{i,j}$ is a HC job) and REFILL will be executed in sequence.

Suppose $J_{i,j}$ starts to execute at t_s and gets preempted at t_e , then before $J_{i,j}$ enters into waiting state again, function RECLAIM($J_{i,j}, BQ, t_s, t_e$) and UPDATE-E+ (only if $J_{i,j}$ is a HC job) is executed in sequence.

Suppose $J_{i,j}$ is a LC job and it starts to execute at t_s and has executed for $\alpha_i \times C_i$ time units at t_e . Then, RECLAIM($J_{i,j}, BQ, t_s, t_e$) is executed before $J_{i,j}$ reverts to its actual deadline.

Figure 6.4: Situations to call functions in MEBA+

Suppose HC job $J_{i,j}$ starts to execute at t_s and $o_{i,j}$ increase to $b_{i,j}$ at t_e . Then, $\text{RECLAIM}(J_{i,j}, BQ, t_s, t_e)$ is executed at t_e . If $o_{i,j}$ is reduced thereafter, then $J_{i,j}$ continues to execute. Otherwise, TRIGGER+ will be executed and the system switches to HC mode.

Similar to Lemma 1, the following lemma presents the mode-switch condition under MEBA+ and EDF-UVD.

Lemma 9 (Mode-switch condition) Suppose a mode-switch to HC mode is triggered at some time instant t^* in a *busy interval* under MEBA+. For each HC task τ_i and each time instant $t \leq t^*$, let $o_{i,t}^m$ denote the maximum amount of O-budget that any job of τ_i has consumed before t in that busy interval. Then, during this busy interval, the system has the following two properties:

Property 1: For all $t \leq t^*$,

$$\sum_{\tau_i \in \tau^H} \frac{o_{i,t}^m}{T_i} \leq \beta^* \times U_H. \quad (6.5)$$

Property 2: t^* is the earliest time instant in that busy interval such that there is a job J_{k,h^*} of some HC task τ_k has $o_{k,h^*} = o_{k,t^*}^m$ but does not signal completion, and

$$\sum_{\tau_i \in \tau^H} \frac{o_{i,t^*}^m}{T_i} = \beta^* \times U_H. \quad (6.6)$$

Proof: Under MEBA, a mode-switch is triggered when some HC job $J_{i,j}$ has consumed $b_{i,j}$ units of budget (i.e., $b_{i,j} = e_{i,j}$) without completion, where $b_{i,j} = T_i \times \left(\beta^* \times U_H - \sum_{\tau_k \in \tau^H \setminus \tau_i} \frac{e_k^m}{T_k} \right)$. On the

other hand, under MEBA+, a mode-switch is triggered when some HC job $J_{i,j}$ has consumed $b_{i,j}$ units of O-budget (i.e., $b_{i,j} = o_{i,j}$) without completion, where $b_{i,j} = T_i \times \left(\beta^* \times U_H - \sum_{\tau_k \in \tau^H \setminus \tau_i} \frac{o_k^m}{T_k} \right)$. Thus, the difference in the mode-switch conditions of MEBA and MEBA+ is that MEBA uses the total consumed budget of each job ($e_{i,j}$) while MEBA+ uses the consumed O-budget of each job ($o_{i,j}$). As a result, the proof of Lemma 7 can be used to prove Lemma 9 except that we need to use $o_i^m/o_{i,j}$ to replace $e_i^m/e_{i,j}$. ■

6.4.3 Implementation Complexity of EDF-UVd and MEBA+

For the static model consisting of n tasks, EDF-VD [16] can be implemented efficiently with a run-time complexity of $O(\log n)$ per event, where an event is either the arrival of a job (event 1), completion of a job (event 2) or the occurrence of mode-switch (event 3).

The priority queue data structure proposed in [66] supports the operations of inserting (insert) and deleting the smallest item (deleteMin) or a specified item (deleteSpec) in logarithmic time, and the operation of finding the smallest item (min) in constant time. In the implementation of EDF-UVd and MEBA+, we maintain two such priority queues Q_L and Q_H . While the system is in LC mode, the scheduler will determine the priority of jobs based on their positions in Q_L . Once the system switches to HC mode, the scheduler will switch to Q_H .

Suppose the system is in LC mode initially, and $J_{x,y}$ denotes the currently executing job. We summarize the operations that MEBA+ with EDF-UVd performs to deal with different events as follows.

1. A new job $J_{i,j}$ arrives at t .
 - (a) Insert $J_{i,j}$ (irrespective of the criticality of τ_i) into both Q_L and Q_H , prioritized according to virtual deadline $d_{i,j}(L)$ and actual deadline $d_{i,j}$, respectively in $O(\log n)$ time.
 - (b) Suppose $J_{x,y}$ is no longer the job with the earliest deadline in Q_L and gets preempted by $J_{i,j}$ (only if $J_{i,j}$ could have earlier deadline than $J_{x,y}$). Then RECLAIM ($O(n)$) and UPDATE_E+ (only for HC job) ($O(1)$) is executed for $J_{x,y}$ in sequence,
 - (c) If $J_{i,j}$ is a HC job, before $J_{i,j}$ starts to execute, COMPUTE_BI+ ($O(n)$) is executed,
2. The currently executing job $J_{x,y}$ completes at t .
 - (a) Then RECLAIM ($O(n)$), UPDATE_E+ (only for HC job) ($O(1)$) and REFILL ($O(1)$) is executed for $J_{x,y}$ in sequence.

- (b) Delete $J_{x,y}$ from Q_L (deleteMin) and Q_H (deleteSpec) in $O(\log n)$ time.
 - (c) Suppose $J_{i,j}$ is the minimum job in Q_L and $J_{i,j}$ is a HC job, then COMPUTE_BI+ ($O(n)$) is executed.
3. The currently executing LC job $J_{x,y}$ has executed for $\alpha_i \times C_i$ time units.
- (a) RECLAIM ($O(n)$) is executed for $J_{x,y}$. Delete $J_{x,y}$ from Q_L (deleteMin) and insert $J_{x,y}$ back to Q_L according to its actual deadline T_i in $O(\log n)$ time.
4. The currently executing job $J_{x,y}$ triggers the potential mode-switch condition.
- (a) RECLAIM ($O(n)$) is executed for $J_{x,y}$.
 - (b) If the value of $o_{x,y}$ is reduced by RECLAIM, the $J_{x,y}$ continues to execute. Otherwise, system switches to HC mode and the scheduler switches to the priority queue Q_H .

As we can observe from the above description, function RECLAIM and COMPUTE_BI+ have a runtime complexity $O(n)$. As a result, MEBA+ (or MEBA) with EDF-UVD can be implemented with a runtime complexity $O(n)$ per event in the worst case. hough MEBA+ strictly dominates MEBA in terms of the ability to postpone mode-switch, it includes additional operations and hence incurs more runtime overhead than MEBA. Thus, if runtime overhead is a concern, one can choose MEBA.

6.5 Schedulability Test

As introduced in Section 6.2.1, each LC task in the dynamic model can receive $\alpha_i \times C_i$ units of budget per job when the system is in HC mode. Thus, α_i denotes the proportion of utilization that is statically reserved for LC task τ_i in HC mode.

Let τ^L denote the subset of LC tasks in τ and α^* denote the service level of all the LC tasks in HC mode, which is equal to the proportion of U_L that is statically reserved for all the LC tasks in HC mode. Given a task set τ characterized by the dynamic model, α^* can be computed as follows.

$$\alpha^* = \frac{\sum_{\tau_i \in \tau^L} \alpha_i \times u_i}{U_L} \quad (6.7)$$

In this section, we derive a sufficient schedulability test for MEBA (and MEBA+) with EDF-UVD. Given a task set τ (and hence the value of α^* is also given), we derive an upper bound of β^* so that τ is

MC-schedulable by MEBA with EDF-UVD. Then, we show that any system that is MC-schedulable by MEBA with EDF-UVD would also be MC-schedulable by MEBA+ with EDF-UVD.

Our approach for deriving the schedulability test is to map a dynamic model based task set τ , scheduled under EDF-UVD and having mode-switch at some instant t^* , to an *equivalent* static model based task set τ' scheduled under EDF-VD. This mapping has the following property. For every feasible job execution sequence $\mathcal{J}$ of job set $\mathbb{J}$ generated by τ with mode-switch at t^* , there is a corresponding feasible job execution sequence $\mathcal{J}'$ of job set $\mathbb{J}'$ generated by τ' , such that there is a deadline miss in $\mathcal{J}$ if and only if there is a deadline miss in $\mathcal{J}'$. Hence the schedulability test of τ' can be used to evaluate the schedulability of τ when it switches mode at t^* . Consider the following task sets τ and τ' whose mapping is shown in Table 6.2.

$$\begin{aligned} \tau &= \{\overbrace{\tau_1, \tau_2, \dots, \tau_p}^{\text{LC Tasks}}, \overbrace{\tau_{p+1}, \tau_{p+2}, \dots, \tau_{p+q}}^{\text{HC Tasks}}\} \mapsto \\ \tau' &= \{\overbrace{\tau'_{1,1}, \tau'_{2,1}, \dots, \tau'_{p,1}}^{\text{HC Tasks}}, \overbrace{\tau'_{1,2}, \tau'_{2,2}, \dots, \tau'_{p,2}}^{\text{LC Tasks}}, \overbrace{\tau'_{p+1}, \tau'_{p+2}, \dots, \tau'_{p+q}}^{\text{HC Tasks}}\} \end{aligned}$$

For a LC task $\tau_i = (T_i, C_i, LC, \alpha_i)$ with virtual deadline $D_i(L) = T_i \times x$, we map it to a HC task $\tau'_{i,1}$ and a LC task $\tau'_{i,2}$. For a HC task $\tau_i = (T_i, C_i, HC)$ with virtual deadline $D_i(L) = T_i \times x$, we map it to a HC task τ'_i with static LC budget $B_i(L) = e_{i,t^*}^m$. Here e_{i,t^*}^m denotes the maximum amount of budget that any job of HC task $\tau_i \in \tau$ can consume up to t^* in the busy interval containing t^* , and this maximum is over all feasible job sequences of τ with mode-switch at t^* .

Mapped Task	T_i	$B_i(L)$	C_i	L_i	$D_i(L)$
$\tau'_{i,1}$	T_i	$\alpha_i \times C_i$	$\alpha_i \times C_i$	HC	$T_i \times x$
$\tau'_{i,2}$	T_i	$(1 - \alpha_i) \times C_i$	$(1 - \alpha_i) \times C_i$	LC	-
τ'_i	T_i	e_{i,t^*}^m	C_i	HC	$T_i \times x$

Table 6.2: Mapped tasks in static MC system τ'

In the following lemma we show that for any feasible job execution sequence $\mathcal{J}$ having mode-switch at t^* , it is feasible to generate an equivalent job sequence $\mathcal{J}'$. Such an equivalent mapping is shown in Figure 6.5 for an example job execution sequence.

Lemma 10 *For each job execution sequence $\mathcal{J}$ of a dynamic task set τ scheduled under MEBA and EDF-UVD and having mode-switch at t^* , it is feasible to generate an equivalent job execution sequence $\mathcal{J}'$ of the static task set τ' defined in Table 6.2 when it is scheduled under EDF-VD.*

Figure 6.5: Mapping $\mathcal{J}$ to $\mathcal{J}'$

Let $\mathbb{J}$ denote the corresponding job set of execution sequence $\mathcal{J}$. Here $\mathcal{J}$ and $\mathcal{J}'$ are equivalent means that for each job $J \in \mathbb{J}$, there is a job $J' \in \mathbb{J}'$ such that J and J' have the same release time, execution requirement, virtual and original deadlines as well as execution schedule (whenever a job J is scheduled, corresponding job J' is scheduled).

Proof: First we consider some simplifications to LC jobs of $\mathbb{J}$. For any job $J \in \mathbb{J}$ released at t such that J is a job of a LC task τ_i , if J executes for more than $\alpha_i \times C_i$ time units in the original schedule under EDF-UVD (say e_i), then we replace it with two jobs J_a and J_b .

1. J_a is released at t , has an execution requirement of $\alpha_i \times C_i$ and deadline at $t + xT_i$.

2. J_b is also released at t , has an execution requirement of $e_i - \alpha_i \times C_i$ and deadline at T_i .

Note that under EDF-UVD, J_b will begin execution only after J_a completes, and hence their combined schedule is identical to that of J .

Consider the following job set in the static model (denoted as $\mathbb{J}'$). For each job $J \in \mathbb{J}$ released at any time instant t , having execution requirement E and deadline t' , a job J' will be released by τ' with the same parameters, such that:

1. If J is a job of HC task τ_i , then static HC task τ'_i will release the job J' .
2. If J is a job of LC task τ_i , $t < t^*$ and J has deadline at $t + T_i$ (job type J_b), then static HC task $\tau'_{i,2}$ will release the job J' .
3. If J is a job of LC task τ_i , and $t \geq t^*$ (job in the HC mode) or J has deadline at $t + xT_i$ (job type J_a), then static HC task $\tau'_{i,1}$ will release the job J' .

Fact 1: It is feasible for τ' to generate $\mathbb{J}'$. *Proof:* The periods of the mapped tasks in the static task set τ' are identical to the periods of the corresponding tasks in the dynamic task set τ . Also, for each job execution requirement E in the dynamic set, the corresponding execution requirement in the static set are no larger than the allocated budgets. ■

Thus, we have shown that for each job $J \in \mathbb{J}$, there is a job $J' \in \mathbb{J}'$ such that J and J' have the same release time, execution requirement, virtual and original deadlines. In the following, we show that they also have the same execution schedule.

Fact 2: The schedule of $\mathbb{J}$ and $\mathbb{J}'$ are identical during $[0, t^*]$. *Proof:* We prove this fact by contradiction. Let us assume the mode-switch occurs at $t'^* (< t^*)$ in $\mathcal{J}'$. That is, some HC job has executed for $B_i(L) = e_{i,t^*}^m$ time units without signalling completion at t'^* , and hence the system is in LC mode only until t'^* .

Observe that the runtime scheduling policy of EDF-UVD and EDF-VD are identical (they both use EDF). Until time instant t'^* , since the jobs of $\mathbb{J}$ and $\mathbb{J}'$ have the same release times, deadlines and **execution requirements**, their schedule is identical.

Since $t'^* < t^*$, by definition of e_{i,t^*}^m , no HC job in $\mathcal{J}$ has executed for e_{i,t^*}^m time units without signalling completion by t'^* . Hence, no job in $\mathcal{J}'$ can execute for e_{i,t^*}^m time units without signalling completion by t'^* . Thus, the assumption that some HC job in $\mathcal{J}'$ has executed for $B_i(L)$ time units without signalling completion at $t'^* (< t^*)$ is false, and hence the system stays in LC mode at least until

t^* even for the execution sequence $\mathcal{J}'$. Since the system stays in LC mode at least until t^* for both $\mathcal{J}'$ and $\mathcal{J}$, the schedule of $\mathbb{J}$ and $\mathbb{J}'$ are identical during $[0, t^*]$. ■

Fact 3: If the mode-switch is triggered at t^* for $\mathcal{J}$, then the mode-switch is also triggered at t^* for $\mathcal{J}'$.

Proof: From **Fact 2** we know that, at t^* , if a job J in $\mathcal{J}$ has remaining execution requirement, then job J' in $\mathcal{J}'$ also has the same amount of remaining execution requirement. Since the mode-switch is triggered by MEBA at t^* , from Lemma 7 we know that at t^* there is an incomplete job J of a HC task $\tau_i \in \tau$ with $E > e_{i,t^*}^m$, where E denotes the total execution requirement of J . Then, we can conclude that J' of HC task $\tau'_i \in \tau'$ also has remaining execution $E' = E > e_{i,t^*}^m = B_i(L)$ at t^* . Hence, the mode-switch is also triggered at t^* for $\mathcal{J}'$. ■

After the mode-switch at t^* , the jobs in $\mathcal{J}$ and $\mathcal{J}'$ have the same requirements (release time, execution, deadline), and they are both scheduled under EDF. Hence, execution sequences $\mathcal{J}$ and $\mathcal{J}'$ are equivalent during the entire busy interval. ■

From Lemma 10 we can infer that there does not exist a dynamic task set τ that is not MC-schedulable under EDF-UVD, but its mapped static task set τ' is MC-schedulable under EDF-VD. This is because if a deadline miss occurs in any feasible job execution sequence $\mathcal{J}$ of τ , it would also occur in the mapped job execution sequence $\mathcal{J}'$ of τ' . Based on this observation, we derive the schedulability test in the following theorem.

Theorem 9 *A dynamic MC task system τ is MC-schedulable by MEBA and EDF-UVD on an uniprocessor platform if the value of β^* satisfies the following condition:*

$$(1 - \alpha^*)(1 - \beta^*) \geq \frac{U_H + U_L - 1}{U_L \times U_H} \Rightarrow \beta \leq 1 - \frac{U_H + U_L - 1}{(U_L \times U_H)(1 - \alpha^*)}, \quad (6.8)$$

and the virtual deadline factor x falls in the following range:

$$\left[\frac{\beta^* \times U_H + \alpha^* \times U_L}{1 - U_L \times (1 - \alpha^*)}, \frac{1 - U_H - \alpha^* \times U_L}{(1 - \alpha^*) \times U_L} \right]$$

Proof: Consider a job execution sequence $\mathcal{J}$ that results in a mode-switch at some time instant t^* . Suppose deadline miss occurs in $\mathcal{J}$. From Lemma 10, we know that for the job sequence $\mathcal{J}$ it is feasible to generate an *equivalent* job sequence $\mathcal{J}'$ by the static system τ' defined in Table 6.2 and scheduled under EDF-VD. Since deadline miss occurs in $\mathcal{J}$, deadline miss also occurs in $\mathcal{J}'$ under EDF-VD. This can be explained as follows. Suppose a job J misses its deadline at some time instant t in $\mathcal{J}$. From Lemma 10, we know that there is a corresponding job J' with the same requirements as J in $\mathcal{J}'$. Since

the schedule is equivalent, J' will also miss its deadline at t . Thus, we can use the schedulability test for task set τ' scheduled under EDF-VD [31] to derive a test for task set τ scheduled under MEBA and EDF-UVD.

From Equation 6.7, we have $\sum_{\tau_i \in \tau^L} \alpha_i \times \frac{C_i}{T_i} = U_L \times \alpha^*$, and hence the total utilization of LC tasks in τ' (tasks of type $\tau'_{i,2}$ in Table 6.2) is $\sum_{\tau_i \in \tau^L} \frac{(1-\alpha_i) \times C_i}{T_i} = U_L \times (1 - \alpha^*)$.

The total utilization of HC tasks in τ' (tasks of type $\tau'_{i,1}$ and τ'_i in Table 6.2) in the HC mode is

$$\sum_{\tau_i \in \tau^H} \frac{C_i}{T_i} + \sum_{\tau_i \in \tau^L} \frac{\alpha_i \times C_i}{T_i} = U_H + U_L \times \alpha^*.$$

From Lemma 7 we have $\sum_{\tau_i \in \tau^H} \frac{e_{i,t^*}^m}{T_i} \leq U_H \times \beta^*$. Thus the total utilization of HC tasks in τ' in the LC mode is

$$\sum_{\tau_i \in \tau^H} \frac{e_{i,t^*}^m}{T_i} + \sum_{\tau_i \in \tau^L} \alpha_i \times \frac{C_i}{T_i} = U_H \times \beta^* + U_L \times \alpha^*.$$

Then, using Theorems 1 and 2 from [31], we get that τ' is MC-Schedulable by EDF-VD on a uniprocessor platform with virtual deadline factor x , if

$$x \leq \frac{1 - U_H - \alpha^* \times U_L}{(1 - \alpha^*) \times U_L} \wedge x \geq \frac{\beta^* \times U_H + \alpha^* \times U_L}{1 - U_L \times (1 - \alpha^*)}$$

and $(1 - \alpha^*)(1 - \beta^*) \geq \frac{U_H + U_L - 1}{U_L \times U_H}.$

This proves the theorem because the above equations are independent of the mode-switch instant t^* . ■

6.5.1 Schedulability Test for MEBA+ and EDF-UVD

In this section, we prove that any system that is MC-schedulable by MEBA and EDF-UVD would also be MC-schedulable by MEBA+ and EDF-UVD. We first use examples to explain the intuition behind the proof for this claim, and then present the formal proof.

Example 6 Let $\tau = \{\tau_1, \tau_2, \tau_3\}$ denote a MC task set, where $\tau_1 = (T_1 = 7, C_1 = 3, LC, \alpha_1 = 0)$, $\tau_2 = (T_2 = 4, C_2 = 2, HC)$ and $\tau_3 = (T_3 = 10, C_3 = 5, HC)$. Figure 6.6 shows the execution sequence $\mathcal{J}$ of job set $\mathbb{J} = \{J_{1,1}, J_{2,1}, J_{2,2}, J_{3,1}\}$, scheduled under MEBA+ and EDF-UVD, where $\beta^* = 0.6$ and the execution requirements of these jobs are $E_{1,1} = 2$, $E_{2,1} = 2$, $E_{2,2} = 2$, $E_{3,1} = 5$.

Figure 6.6: Execution Sequence with Budget Reclamation

In this execution sequence, $J_{3,1}$ misses its deadline at $t = 10$, and the earliest budget reclamation happens between $J_{2,2}$ and $J_{1,1}$ during $[4, 5]$. We now show that there exists another job sequence that will also miss its deadline under MEBA and EDF-UVD.

Suppose we update the execution requirements of jobs in $\mathbb{J}$ as follows. $\hat{E}_{1,1} \leftarrow E_{1,1} + (5 - 4) = 3$ and $\hat{E}_{2,2} \leftarrow E_{2,2} - (5 - 4) = 1$. Let us denote the resulting job sequence as $\hat{\mathcal{J}}^3$. Then, in the updated job execution sequence shown in Figure 6.7, $\hat{\mathcal{J}}$ has the following properties:

1. The system still stays in LC mode up to 5 time units in $\hat{\mathcal{J}}$.
2. $\hat{\mathcal{J}}$ and $\mathcal{J}$ are identical after 5.
3. $\hat{o}_{2,2} = o_{2,2} = 1$, $\hat{e}_{2,2} = e_{2,2} - (5 - 4) = 1$, $\hat{o}_{1,1} = o_{1,1} + (5 - 4) = 3$ and $\hat{e}_{1,1} = e_{1,1} + (5 - 4) = 3$.

As shown in the example, we can transform a job execution sequence with budget reclamation into an equivalent one without budget reclamation by adjusting the job execution requirements. Therefore, if there is a deadline miss in the job execution sequence under MEBA+, then there will also be a deadline miss in the transformed job execution sequence under MEBA. We now use this intuition to formally prove the schedulability equivalence between MEBA+ and MEBA.

³We use $\hat{\cdot}$ to differentiate the parameters of $\hat{\mathcal{J}}$ from the parameters of $\mathcal{J}$.

Figure 6.7: Execution Sequence after Updating Execution Requirement

Lemma 11 Let $\mathbb{J}$ denote any job set released by task system τ , and $\mathcal{J}$ denote the execution sequence of $\mathbb{J}$ scheduled by MEBA+ and EDF-UVd during a single busy interval $[0, t_f]$. Suppose the earliest budget reclamation in $\mathcal{J}$ happens between $J_{x,y}$ and $J_{a,b}$ during $[t_s^1, t_e^1]$, and the budget consumed by $J_{a,b}$ during $[t_s^1, t_e^1]$ is assumed to be the U-budget of $J_{x,y}$.

Suppose we consider another job set $\hat{\mathbb{J}}$ of task system τ that is identical to $\mathbb{J}$, except: $\hat{E}_{x,y} \leftarrow E_{x,y} + t_e^1 - t_s^1$ and $\hat{E}_{a,b} \leftarrow E_{a,b} - (t_e^1 - t_s^1)$. Let $\hat{\mathcal{J}}$ denote the execution sequence of $\hat{\mathbb{J}}$ scheduled by MEBA+ and EDF-UVd. Then, $\hat{\mathcal{J}}$ has the following properties:

1. The system still stays in LC mode up to t_e^1 in $\hat{\mathcal{J}}$.
2. $\hat{\mathcal{J}}$ and $\mathcal{J}$ are identical after t_e^1 .
3. The mapped jobs $\hat{J}_{a,b}$ and $\hat{J}_{x,y}$ satisfy the following. $\hat{o}_{a,b} = o_{a,b}$, $\hat{e}_{a,b} = e_{a,b} - (t_e^1 - t_s^1)$, $\hat{o}_{x,y} = o_{x,y} + (t_e^1 - t_s^1)$ and $\hat{e}_{x,y} = e_{x,y} + (t_e^1 - t_s^1)$.

Proof: In $\mathcal{J}$, the earliest budget reclamation happens between $J_{x,y}$ and $J_{a,b}$, and the budget consumed by $J_{a,b}$ during $[t_s^1, t_e^1]$ is assumed to be the U-budget of $J_{x,y}$. Then we can infer that:

1. There is no U-budget from jobs with higher priority than $J_{x,y}$ during $[t_s^1, t_e^1]$.

2. $J_{a,b}$ is the active job with the highest priority during $[t_s^1, t_e^1]$.

Fact 1: $\hat{\mathbb{J}}$ is a **valid job set** of τ . *Proof:* Since $J_{x,y}$ has at least $t_e^1 - t_s^1$ units of U-budget when it completes and $J_{x,y}$ does not consume any U-budget, then we have $o_{x,y} = e_{x,y} = E_{x,y}$, and $o_{x,t_s^1}^m - o_{x,y} \geq t_e^1 - t_s^1$ (if $J_{x,y}$ is a HC job) or $B_x(L)(= C_x) - o_{x,y} \geq t_e^1 - t_s^1$ (if $J_{x,y}$ is a LC job). Thus, using the fact $o_x^m \leq C_x$, we can infer that

$$C_x - o_{x,y} \geq t_e^1 - t_s^1 \Rightarrow \hat{E}_{x,y} = E_{x,y} + t_e^1 - t_s^1 \leq C_x$$

, which means $\hat{J}_{x,y}$ is a valid job of τ_x . On the other hand, since $E_{a,b} \leq C_a \Rightarrow \hat{E}_{a,b} = E_{a,b} - (t_e^1 - t_s^1) \leq C_a$, $\hat{J}_{a,b}$ is also a valid job of τ_a . Thus, $\hat{\mathbb{J}}$ is a **valid job set** of τ . ■

Fact 2: The system stays in LC mode up to t_e^1 in $\hat{\mathcal{J}}$. *Proof:*

Note that jobs in $\hat{\mathbb{J}}$ and $\mathbb{J}$ are exactly the same (release times, deadlines and hence priorities) except $\hat{E}_{x,y} \leftarrow E_{x,y} + t_e^1 - t_s^1$ and $\hat{E}_{a,b} \leftarrow E_{a,b} - (t_e^1 - t_s^1)$. Suppose $J_{x,y}$ completes at $t_y (< t_s^1)$ after executing for $E_{x,y}$ units. Then, $\hat{\mathcal{J}}$ and $\mathcal{J}$ are identical during $[0, t_y]$ and $\hat{J}_{x,y}$ will also execute for $E_{x,y}$ units by t_y . In $\mathcal{J}$, any job executed during $[t_y, t_s^1]$ must have higher priority than $J_{x,y}$ because otherwise first budget reclamation should occur before t_s^1 . Thus, in $\hat{\mathcal{J}}$, $\hat{J}_{x,y}$ cannot execute during $[t_y, t_s^1]$ and still has an execution requirement of $t_e^1 - t_s^1$ at t_s^1 . This also implies that $\hat{\mathcal{J}}$ and $\mathcal{J}$ are identical during $[0, t_s^1]$.

As a result, $\hat{J}_{x,y}$ is the active job with highest priority during $[t_s^1, t_e^1]$ in $\hat{\mathcal{J}}$ as long as the system stays in LC mode during $[t_s^1, t_e^1]$. This implies that, **if a mode-switch is triggered during $[t_s^1, t_e^1]$, it could only be triggered by $\hat{J}_{x,y}$ and $\hat{J}_{x,y}$ must be a HC job.**

Since $J_{x,y}$ has at least $t_e^1 - t_s^1$ units of U-budget at t_s^1 in $\mathcal{J}$, then we have $o_{x,t_s^1}^m - o_{x,y} \geq t_e^1 - t_s^1$. Suppose the mode-switch is triggered at $t^* \in (t_s^1, t_e^1)$ in $\hat{\mathcal{J}}$. Then at t^* , $\hat{J}_{x,y}$ satisfies the following condition.

$$\left(\overbrace{\hat{o}_{x,y} = \hat{o}_{x,t^*}^m = o_{x,y} + (t^* - t_s^1)}^{\text{from Lemma 9}} \right) \wedge \left(o_{x,t_s^1}^m \geq o_{x,y} + t_e^1 - t_s^1 \right) \Rightarrow \underline{\hat{o}_{x,t^*}^m < o_{x,t_s^1}^m}$$

However, this contradicts the fact that $\hat{o}_{x,t^*}^m \geq \hat{o}_{x,t_s^1}^m = o_{x,t_s^1}^m$ because $\mathcal{J}$ and $\hat{\mathcal{J}}$ are identical during $[0, t_s^1]$, i.e., $\hat{o}_{x,t_s^1}^m = o_{x,t_s^1}^m$ and $t^* > t_s^1 \Rightarrow \hat{o}_{x,t^*}^m \geq \hat{o}_{x,t_s^1}^m$. Hence, **the system stays in LC mode up to t_e^1 in $\hat{\mathcal{J}}$.** ■

Fact 3: $\hat{o}_{x,y} = o_{x,y} + (t_e^1 - t_s^1)$ and $\hat{e}_{x,y} = e_{x,y} + (t_e^1 - t_s^1)$. *Proof:* From **Fact 2** we know that the system stays in LC mode up to t_e^1 in $\hat{\mathcal{J}}$, and there is no available U-budget with higher priority. Thus, $\hat{J}_{x,y}$ will execute during $[t_s^1, t_e^1]$ using its O-budget to meet the increased execution requirement.

Then, no budget reclamation happens during $[t_s^1, t_e^1]$ in $\hat{\mathcal{J}}$ and $\hat{J}_{x,y}$ completes at t_e^1 . As a result, we have $\hat{o}_{x,y} = o_{x,y} + (t_e^1 - t_s^1)$ and $\hat{e}_{x,y} = e_{x,y} + (t_e^1 - t_s^1)$. ■

Fact 4: $\hat{\mathcal{J}}$ and $\mathcal{J}$ are also identical during $[t_e^1, t_f]$ and $\hat{o}_{a,b} = o_{a,b}$ and $\hat{e}_{a,b} = e_{a,b} - (t_e^1 - t_s^1)$. *Proof:* During $[t_e^1, t_f]$, all jobs in $\hat{\mathbb{J}}$ and $\mathbb{J}$ have the same release times, deadlines and remaining execution demand. As a result, $\hat{\mathcal{J}}$ and $\mathcal{J}$ are also identical during $[t_e^1, t_f]$. Since $\hat{\mathcal{J}}$ and $\mathcal{J}$ are also identical during $[t_e^1, t_f]$, and $\hat{E}_{a,b} = E_{a,b} - (t_e^1 - t_s^1)$, we have $\hat{o}_{a,b} = o_{a,b}$ and $\hat{e}_{a,b} = e_{a,b} - (t_e^1 - t_s^1)$. ■

Facts 1-4 thus prove the lemma. ■

Figure 6.8: Execution Sequence without Budget Reclamation

If we apply Lemma 11 recursively on the earliest budget reclamation interval in $\mathcal{J}$ of Example 6 (i.e., $[4, 5]$ and $[6, 7]$), then the execution requirements of jobs in Example 6 are updated as follows.

$$\hat{E}_{1,1} \leftarrow E_{1,1} + 1 = 3, \hat{E}_{2,2} \leftarrow E_{2,2} + 1 - 1 = 2, \hat{E}_{3,1} \leftarrow E_{3,1} - 1 = 4$$

As shown in Figure 6.8, $\hat{\mathcal{J}}$ is the execution sequence of the new job set $\hat{\mathbb{J}}$ when it is scheduled by MEBA+ and EDF-UVD. As we can observe, no budget reclamation happens in $\hat{\mathcal{J}}$, and hence $\hat{\mathcal{J}}$ is also the execution sequence of job set $\hat{\mathbb{J}}$ when it is scheduled by MEBA and EDF-UVD. In $\hat{\mathcal{J}}$, the deadline miss still happens at $t = 10$. By creating such an equivalent schedule, we can prove that if a task system is MC-schedulable under MEBA and EDF-UVD, then it is also MC-schedulable under MEBA+ and

EDF-UVD. Based on this observation, we present the following theorem.

Theorem 10 *Suppose a MC system τ is MC-schedulable under MEBA and EDF-UVD. Then, it is also MC-schedulable under MEBA+ and EDF-UVD.*

Proof: Suppose τ is MC-schedulable under MEBA and EDF-UVD, but not MC-schedulable under MEBA+ and EDF-UVD. Let $\mathcal{J}$ denote an execution sequence of a job set $\mathbb{J}$ of τ scheduled under MEBA+ and EDF-UVD during a busy interval $[0, t_f]$, where the earliest deadline miss occurs for a job $J_{h,k}$ at time instant t_f . Note, by definition, there is at most one mode-switch in $\mathcal{J}$.

In the schedule $\mathcal{J}$, let $\phi_{i,j}$ denote the amount of U-budget any job $J_{i,j}$ has consumed and $\varphi_{i,j}$ denote the amount of $J_{i,j}$'s U-budget consumed by other jobs in $\mathcal{J}$. By using Lemma 11 recursively on the earliest budget reclamation interval in $\mathcal{J}$,

we finally get a new job execution sequence $\hat{\mathcal{J}}$ such that any job $\hat{J}_{i,j}$ in $\hat{\mathcal{J}}$ satisfies the following.

1. $\hat{E}_{i,j} = E_{i,j} - \phi_{i,j} + \varphi_{i,j}$, i.e., the amount of the U-budget consumed by $J_{i,j}$ is reduced from $E_{i,j}$ and the amount of $J_{i,j}$'s U-budget consumed by other jobs is added to $E_{i,j}$.
2. $\hat{e}_{i,j} = \hat{o}_{i,j} = e_{i,j} - \phi_{i,j} + \varphi_{i,j}$ (from property 3 of Lemma 11).

The second property above implies that no budget reclamation happens in $\hat{\mathcal{J}}$. Hence, $\hat{\mathcal{J}}$ is also the execution sequence of $\mathbb{J}$ when it is scheduled under MEBA and EDF-UVD. Since $\mathbb{J}$ is MC-schedulable under MEBA and EDF-UVD, $\hat{J}_{h,k}$ must meet its deadline and hence we have $\hat{E}_{h,k} = \hat{o}_{h,k} = \hat{e}_{h,k}$. Since $\hat{E}_{h,k} = E_{h,k} - \phi_{h,k} + \varphi_{h,k}$ and $\hat{e}_{h,k} = e_{h,k} - \phi_{h,k} + \varphi_{h,k}$, we have $E_{h,k} = e_{h,k}$. However, this contradicts the condition $E_{h,k} > e_{h,k}$, because $J_{h,k}$ misses its deadline at t_f in $\mathcal{J}$. Therefore the assumption is false and this proves the theorem. ■

6.6 Trade-off between α^* and β^*

From Theorem 9 in Section 6.5 we know that a dynamic system τ is MC-Schedulable under MEBA (MEBA+) and EDF-UVD if

$$(1 - \alpha^*)(1 - \beta^*) \geq M \text{ where } M = \frac{U_H + U_L - 1}{U_L \times U_H}$$

Thus, there is a trade-off between β^* and α^* . In this section, we discuss the influence of the values of β^* and α^* (assuming we are able to set α_i for each LC task) on the system behavior.

A higher value for β^* implies a higher total budget allocation for HC tasks in the LC mode, and consequently the mode-switch is less likely to be triggered. It is then possible that LC tasks are able to receive up to C_i units of budget in the LC mode for a longer duration of time. On the other hand, a higher value for α^* implies higher service level for LC tasks in the HC mode, but then the resulting smaller value for β^* may lead to frequent occurrence of mode-switch. Thus, β^* controls the amount of time for which LC tasks receive full service, whereas α^* controls the minimum guaranteed service for LC tasks at all times.

The likelihood that the system switches to HC mode minimizes when $\alpha^* = 0$ and $\beta^* (= 1 - M)$. Alternatively, if we wish to support LC tasks as much as possible, while ensuring that the likelihood of mode-switch is relatively small, then we can conduct large-scale simulations of the system and monitor the value of $\frac{\sum_{\tau_i \in \tau^H} e_i^m / T_i}{U_H}$ in different busy intervals. Suppose β_{max} is the maximum observed value of $\frac{\sum_{\tau_i \in \tau^H} e_i^m / T_i}{U_H}$ in these simulations, then we can set $\beta^* = \beta_{max}$. Assuming the execution requirements of HC tasks are well-represented in these simulations, the likelihood that the system triggers a mode-switch would be low. To further avoid mode-switch, we can set $\beta^* = F \cdot \beta_{max}$ where $F > 1$.

Note that the value of β^* is upper bounded by $1 - M$. Therefore, if τ has a large $U_H + U_L$ value (small $1 - M$) and $1 - M \ll \beta_{max}$, then the overall demand of HC tasks will frequently exceed $(1 - M)U_H$. As a result, the system is likely to switch to HC mode. To support LC tasks as much as possible in this case, it might be better to set $\beta^* = 0$ and $\alpha^* = 1 - M$.

HC budget allocation for LC tasks. The technique to allow LC tasks to receive degraded budget in HC mode is mainly designed for non-critical QoS applications such as multimedia applications. The application designer could provide a set of possible HC budgets ($B_i(H1), B_i(H2), \dots$) for the application, and the system designer can choose among them based on the schedulability condition. Thus, once the system switches to HC mode, the application could switch to another version of code with a lower budget requirement and service quality.

6.6.1 Total System Utilization

When the system is in the LC mode, each LC task τ_i receives a budget of C_i and all HC tasks combined receive a total budget proportional to $\beta^* \times U_H$. Thus the system utilization in the LC mode can be defined as follows.

$$SU_L = \beta^* \times U_H + \sum_{\tau_i \in \tau^L} \frac{C_i}{T_i} = \beta^* \times U_H + U_L$$

Figure 6.9: β^* and weight w when $SU(w)$ is maximized

Similarly, when the system is in the HC mode, LC tasks receive a budget of $B_i(H) = \alpha_i \times C_i$ and HC tasks receive a budget of C_i per job. Thus the system utilization in the HC mode can be defined as follows.

$$SU_H = \sum_{\tau_i \in \tau^L} \frac{B_i(H)}{T_i} + U_H = \alpha^* \times U_L + U_H$$

Consider the following definition of *total system utilization*, where $w \in [0, 1]$

$$\begin{aligned} SU(w) &= w \times SU_L + (1 - w) \times SU_H \\ &\leq U_H(1 - w + w\beta^*) + U_L(w + (1 - w)(1 - \frac{M}{1 - \beta^*})) \\ &= U_H(1 - w) + U_L + U_H w\beta^* - \frac{U_L(1 - w)M}{1 - \beta^*} \end{aligned} \tag{6.9}$$

$SU(w)$ gives a value for the total system utilization, assuming a weight of w for the LC mode

Figure 6.10: Case: $U_H = 0.8$ and $U_L = 0.7$

utilization. Since $SU(w)$ is a convex function of β^* and β^* is in the range $[0, 1 - M]$, by taking the first derivate and equating it to 0, $SU(w)$ is maximized when

$$\beta^* = \max \left\{ 0, \min \left\{ 1 - M, 1 - \sqrt{\frac{M(1-w)U_L}{wU_H}} \right\} \right\} \quad (6.10)$$

We performed some experiments to understand the impact of weight w on the total system utilization. In Figure 6.9 we show how the value of β^* varies when $SU(w)$ is maximized. We fix $U_{sum} = U_H + U_L = 1.5$ and assign different values to U_L and w . Z-axis plots the value of β^* for which $SU(w)$ is maximized, x-axis denotes U_L , and y-axis denotes weight w . $U_L \in \{U_{sum} - 1, U_{sum} - 0.9, U_{sum} - 0.8, \dots, 1\}$ and $w \in \{0.02, 0.04, 0.06, \dots, 1\}$ in the figure.

As we can observe, in many cases, the value of β^* for which $SU(w)$ is maximized is not sensitive to the value of w . The reason for this observation is that when the system has a large U_{sum} , the value of $1 - M$ is relatively small. For such systems, $SU(w)$ is maximized when either $\beta^* = 1 - M$ or $\beta^* = 0$ for almost all the values of w .

To provide further insight into this observation, in Figure 6.10 we plot these values of β^* as a function

of weight w for a specific value of $U_L (= 0.7)$ and $U_H (= 0.8)$. We can observe that $\beta^* = 0$ when $w \in [0, \approx 0.42]$ and is equal to $1 - M (= 0.107)$ when $w \in [\approx 0.5, 1]$.

6.7 Evaluation

An important advantage of the dynamic model is that the designer is not required to specify LC budgets for individual HC tasks. There are two other MC models that do not require the designer to provide such budgets. In the worst-case reservations model commonly used in the safety-critical industry today (e.g., European automotive industry (AUTOSAR)⁴ and in the avionics domain (ARINC)⁵), tasks are always guaranteed to receive C_i units of budget per job, irrespective of their criticality level. The dynamic model generalizes this model, and they are equivalent if we set $\beta^* = 1$. Although this model ensures isolation between tasks at different criticality levels, it does not allow for efficient sharing of the processor. Another one is the elastic model [52]. In the elastic model [52], all HC tasks receive C_i units of budget per job, and LC tasks have a maximum period T_i^{max} and a set of early-release points which enable them to release more frequently. Thus, LC tasks are only guaranteed to receive C_i units of budget every T_i^{max} time units, and the service level of LC tasks is fixed at $(1 - U_H)/U_L$. The dynamic model also generalizes this model, and they are equivalent if we set $\beta^* = 0$ and $\alpha^* = (1 - U_H)/U_L$. In the remainder of this section, we evaluate the dynamic model from three different aspects.

6.7.1 Minimum Guaranteed Service for LC Tasks

In this section we experimentally evaluate the performance of the dynamic model in terms of its ability to provide minimum guaranteed service to LC tasks. We compare its performance to the service adaption strategy [19], which also uses a EDF-based scheduling policy and provides a minimum guaranteed service to LC tasks. This strategy uses EDF-VD and decreases the dispatch frequency of LC tasks when the system switches mode. It finds the minimal possible factor y (single value) to extend the periods of all LC tasks i.e., $T_i \leftarrow y \times T_i$, where $y \geq 1$. Therefore, the minimum guaranteed service for LC tasks that the service adaption strategy can support is equal to $\frac{1}{y}$. We compare this value with α^* in the dynamic model to evaluate their relative performance.

For this comparison, we have to use task systems characterized by the static model (i.e., given $C_i(L)$ value for each HC task τ_i), because the service adaptation strategy is designed for such systems. To

⁴<http://www.autosar.org/>

⁵<http://www.arinc.com/>

ensure a fair comparison, we assume that β^* in the dynamic model is set to a value such that $\beta^* \times U_H = \sum_{\tau_i \in \tau^H} C_i(L)/T_i$. We then compute the maximum possible α^* that still guarantees schedulability.

The task set generation procedure can be summarized as follows. Each task is generated based on the following parameters.

- Task τ_i is a HC task with probability $PH = 0.5$.
- $C_i(L)$ is drawn using a uniform distribution over $[1, 10]$.
- C_i is drawn using a uniform distribution over $[C_i(L), RC \times C_i(L)]$, where $RC \in \{3, 4, 5\}$.
- T_i is drawn using a uniform distribution over $[C_i, 200]$.

Task set τ is empty initially. Randomly generated tasks based on the above procedure are added to the task set repeatedly. Then, $U_A = \frac{U(L)+U(H)}{2}$ is equal to the average utilization of τ in LC and HC mode at any point in the generation process (computed based on tasks that have already been added to τ). We classify τ based on the range in which U_A lies: $[0.54, 0.55]$, $[0.59, 0.60]$, $[0.64, 0.65]$, $[0.69, 0.7]$, $[0.74, 0.75]$. We do not consider the case when $U_A > 0.75$, because very few task sets with $U_A > 0.75$ are schedulable under the service adaption strategy. A new task is added to τ until U_A falls in the range we choose. However if U_A becomes greater than the upper bound of the range, we discard the entire task set and repeat the process.

In Figure 6.11, we show the average minimum guaranteed service for LC tasks that the dynamic model and service adaption strategy can support when $RC = 3, 4$ and 5 . Each value in the figure is based on 1000 task sets. If a task set is not schedulable by either the dynamic model or the service adaption strategy, then the service level of LC tasks is assumed to be 0. This is reasonable because there is no guarantee on deadlines for such task sets. Even though both the service adaption strategy and the dynamic model are based on EDF, the dynamic model always outperforms the service adaption strategy. For some settings, its performance is almost two times better than the performance of the latter. Figure 6.12 shows the average minimum guaranteed service when $RC = 4$ and $PH = 0.25, 0.5, 0.75$ respectively. Here PH denotes the probability that a task is a HC task. As can be seen in the figures, the dynamic model significantly outperforms the service adaptation strategy for all these different settings.

The reasons resulting in this performance gap can be summarized as follows:

1. the schedulability analysis for service adaption strategy is based on pessimistic approximate demand bound functions whereas for the dynamic model it is based on utilization-based tests, and

Figure 6.11: Average minimum guaranteed service for LC tasks

2. we set virtual deadlines for LC tasks to reduce the scenario where jobs of LC tasks receive unnecessary budget (advantage of EDF-UVD over EDF-VD).

Another important advantage of the dynamic model is that it provides the flexibility to set different HC service levels for each LC task (α^* is split among the different LC tasks based on α_i), while the service adaption strategy has to decrease the dispatch frequency of all the LC tasks to the same degree.

6.7.2 Analysis of Probability of Mode-Switch

Under the MEBA strategy of dynamic model, the mode-switch is triggered only when the total allocated LC budget for all HC tasks combined exceeds $\beta^* \times U_H$. This is different from the static model in which a mode-switch is triggered even when a single HC task exceeds its allocated budget. Therefore,

Figure 6.12: Average HC service level for LC tasks with varying PH

the dynamic model can reduce the probability that a task system switches mode when compared to the static model. In this section we analytically compare the two models from this aspect. We first use a simple example to illustrate how the dynamic model can reduce the likelihood of mode-switch.

Example 7 Suppose we have a task system $\tau = \{\tau_1, \tau_2, \tau_3\}$ as given in Table 6.3.

Table 6.3: Example Task Set

Task	T_i	C_i	L_i	α_i
τ_1	10	5	LC	0
τ_2	10	4	HC	
τ_3	10	4	HC	

Using Equation 6.8, we set $\beta^* = 0.25$. Thus, as long as $(e_1^m/10 + e_2^m/10)/0.8 \leq 0.25$, i.e., $e_1^m + e_2^m \leq 2$, the system would stay in the LC mode (from Lemma 7). Instead, if we have a fixed budget

as in the static model, e.g., $B_1(L) = B_2(L) = 1$, the system would switch to HC mode when either τ_2 or τ_3 executes beyond 1 time unit. Thus, in the dynamic model, even if τ_2 executes for 1.05 time units, the system could stay in the LC mode as long as no job of τ_3 has executed beyond 0.95 time units.

Let $P(s_i)$ denote the probability that no job of τ_i executes for more than $C_i \times s_i$ time units in a certain busy interval, where s_i is a random variable in the range $(0, 1]$. Then, a task set τ does not switch mode under the static model only when no HC task executes beyond its fixed LC budget $B_i(L) = C_i(L)$ determined offline (let $C_i(L) = \beta_i \times C_i$ for each i). Then, assuming all tasks are independent, the probability that the system does not switch to HC mode is

$$P_{noswitch}^s = \prod_{\tau_i \in \tau^H} P(s_i = \beta_i) \quad (6.11)$$

On the other hand, according to Lemma 7, the system does not switch mode under MEBA as long as

$$\sum_{\tau_i \in \tau^H} \frac{e_{i,t}^m}{T_i} \leq \beta^* \times U_H.$$

Suppose $e_{i,t}^m = s_i \times C_i$. Then, the probability that the system does not switch to HC mode under MEBA is

$$P_{noswitch}^d = \sum_{\tau_i \in \tau^H} \sum_{\substack{s_i \in (0,1] \\ \frac{s_i \times C_i}{T_i} \leq \beta^* \times U_H}} \left(\prod_{\tau_i \in \tau^H} P(s_i) \right) \quad (6.12)$$

We can see that $P_{noswitch}^d \geq P_{noswitch}^s$, because $s_i = \beta_i$ is one possible assignment that satisfies

$$\sum_{\tau_i \in \tau^H} \frac{s_i \times C_i}{T_i} \leq \beta^* \times U_H$$

(using Lemma 8), but there exists many more assignments of s_i that also satisfy the above condition. Further, we can also observe that the probability of mode-switch under the static model increases exponentially with the number of HC tasks, while MEBA with the dynamic model can mitigate this problem.

Table 6.4: Example distribution for $P(s_i)$

s_i	0.1	0.2	0.3	0.4	0.5
$P(s_i)$	0.01	0.05	0.2	0.5	0.8
s_i	0.6	0.7	0.8	0.9	1.0
$P(s_i)$	0.9	0.95	0.98	0.995	1.0

From Equations 6.11 and 6.12, we can evaluate the performance of dynamic and static models ana-

lytically in terms of their ability to reduce the mode-switch probability. Now we plot Equation 6.11 and 6.12 using a specific distribution for $P(s_i)$. We set $\beta^* = 1 - M = (1 - U_L)(1 - U_H)/(U_L \cdot U_H)$ and $\alpha^* = 0$ in the dynamic model because this choice postpones the mode-switch as much as possible. For the static model, to ensure a fair comparison, we set $\beta_i = 1 - M \Leftarrow C_i(L) = (1 - M) \times C_i$ for each HC task τ_i . Thus, both the models have the same LC system service level for HC tasks (β^*). But the dynamic model uses the runtime strategy MEBA to distribute this service among the HC tasks, while the static model uses $\beta_i \times C_i$ as a fixed LC budget.

Figure 6.13: Comparison of the probability that the mode-switch does not happen

Suppose τ comprises n HC tasks and $P(s_i)$ conforms to the distribution shown in Table 6.4 for each task. Then we can calculate $P_{noswitch}^s$ and $P_{noswitch}^d$ when $n \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. Figure 6.13 plots Equations 6.11 and 6.12 as a function of the number of HC tasks, where x-axis denotes the number of HC tasks and y-axis denotes the probabilities. The mode-switch probability depends on the value of $\beta^* = 1 - M$. Hence, we show the results for $1 - M \in \{0.45, 0.55, 0.65, 0.75\}$. When $1 - M > 0.75 \vee 1 - M < 0.45$, the probability would either approach 1 or 0 according to the distribution in Table 6.4. Hence we do not present the results for $1 - M > 0.75 \vee 1 - M < 0.45$.

We can see that when $1 - M$ has a large value, the mode-switch probability for both dynamic and static models is low, and vice versa. Also, the dynamic model can significantly reduce this probability when compared to the static model under different settings. We can observe that as the number of tasks increase, $P_{noswitch}^s$ decreases monotonically, while the value of $P_{noswitch}^d$ is stable.

Note that the mode-switch probability for the static model that we computed is in fact the probability of mode-switch in several MC studies (e.g., [17, 19, 22–24, 29–31, 59, 60]). Other works (e.g., [46, 47, 58, 67]) have different levels of mode-switch depending on the number of HC tasks that are executing beyond their LC budgets; in these studies the first mode-switch has a probability identical to what we computed for the static model. Hence, the results of Figure 6.13 provide a direct comparison between the dynamic model and all the above studies.

6.7.3 Simulation Results about Mode-Switch

In the previous section, we compare the probability of mode-switch in dynamic and static models analytically. In this section, we evaluate 1) MEBA+ (with EDF-UVD), 2) MEBA (with EDF-UVD) and 3) static LC budget allocation (with EDF-VD) through simulations.

Task Set Generator: In the simulation, we use the same task set generator as described in Section 6.7.1 to generate MC task sets. We generate task sets with $U_A \in \{0.7, 0.8\}$ and $RC \in \{3, 5\}$. For each setting, we choose 5000 task sets that are MC-schedulable by both EDF-VD and EDF-UVD. For fair comparison, in the simulation of MEBA+ or MEBA (with EDF-UVD), we set

$$\beta^* = \frac{1}{U_H} \sum_{\tau_i \in \tau^H} C_i(L)/T_i.$$

Simulation Process: In the simulation, each task will release a job every T_i time units from time instant $t = 0$ to $t = 10000$. To ensure a fair comparison with static LC budget allocation (with EDF-VD) and focus on how much the mode-switch can be postponed, LC task in the dynamic model has $\alpha_i = 0$.

At runtime, each job of τ_i will require $\lfloor s_i \times C_i \rfloor$ time units of execution. The probability that J_i requires $\lfloor s_i \times C_i \rfloor$ time units of execution is equal to $P(s_i)$, where $s_i \in \{0.1, 0.2, \dots, 1.0\}$ and $P(s_i)$ conforms to binomial distribution $B(10, 0.4)$ or $B(10, 0.6)$ as shown in Figure 6.14. To ensure a fair comparison, a list of execution requirements $\{E_{i,1}, E_{i,2}, \dots\}$ are generated off-line according to the distributions of $P(s_i)$. As a result, in the simulation of MEBA+ (with EDF-UVD), MEBA (with EDF-

UVD) or static LC budget allocation (with EDF-VD), the j_{th} job of τ_i would require $E_{i,j}$ units of budget.

Figure 6.14: $P(s_i) \sim B(10, 0.4)$ and $P(s_i) \sim B(10, 0.6)$

Performance Metric: We focus on comparing 1) MEBA+ (with EDF-UVD), 2) MEBA (with EDF-UVD) and 3) static LC budget allocation (with EDF-VD) using the following two metrics.

- Metric 1: The number of mode-switches that occur during $[0, 10000]$.
- Metric 2: The percentage of time that the system stays in LC mode during $[0, 10000]$.

Intuitively, metric 2 is negatively related to metric 1. If the number of mode-switches that occur during $[0, 10000]$ increases, the system would switch to HC mode more frequently, and hence stays in LC mode for less time. However, we should note that, the service level of LC tasks in HC mode would influence both metric 1 and 2. If LC jobs receive guaranteed budgets, the system is less likely to be idle. As a

result, the time instant to revert to LC mode is postponed and the system stays in HC mode for longer time. In our case, no LC jobs would execute in HC mode. Thus, the percentage of time that the system stays in LC mode is still relatively high ($> 50\%$) even when a lot of mode-switches are triggered.

Figures 6.15 to 6.22 show the simulation results of the three strategies using task sets with $U_A \in \{0.7, 0.8\}$, $RC \in \{3, 5\}$ and $P(s_i) \sim B(10, 0.4)$ or $B(10, 0.6)$. In these figures, the left y-axis denotes the percentage of time that the system stays in LC mode and the right y-axis denotes the number of mode-switches occurred in $[0, 10000]$. The x-axis of those figures denotes the number of HC tasks (n_H) in a task set.

As we can observe from the simulation results, the MEBA (or MEBA+) with EDF-UVD dominates the static LC budget allocation with EDF-VD under different settings. The performance gap between MEBA (or MEBA+) and static LC budget allocation also enlarges as n_H increases. This trend is in line with our analysis in Section 6.7.2. When the value of n_H reaches 8 or 9, very few mode-switches are triggered in the simulation of MEBA (or MEBA+) with EDF-UVD.

Besides, for MEBA, MEBA+ and static LC budget allocation, more mode-switches are triggered when $P(s_i) \sim B(10, 0.6)$ compared to the case when $P(s_i) \sim B(10, 0.4)$. When $P(s_i) \sim B(10, 0.6)$, each job is more likely to have a large execution requirement, and hence is more likely to trigger the mode-switch. As a result, the system also stays in LC mode for a longer time when $P(s_i) \sim B(10, 0.4)$.

As we can observe from Figures 6.15 to 6.22, MEBA+ strictly dominates MEBA for different settings. Even though the absolute performance gap between MEBA+ and MEBA is not so large as the gap between MEBA (or MEBA+) and static LC budget allocation strategy, the relative improvement of MEBA+ is still considerable when the value of n_H is large. Table 6.5 shows the percentage of number of mode-switches that MEBA+ can reduce compared to MEBA for $n_H \in \{5, 6, 7, 8, 9\}$. As we can observe from Table 6.5, a large percentage of mode-switches are avoided by MEBA+. For example, when $n_H = 9$, $P(s_i) \sim B(10, 0.4)$, $RC = 3$ and $U_A = 0.7$, 54.2% of the mode-switches that occurred in MEBA are avoided by MEBA+.

6.7.4 Total System Utilization

We introduced a technique to maximize the total system utilization in Section 6.6. Here we compare the maximum total system utilization that dynamic and static models can support for different values of U_H and U_L . For the dynamic model this parameter can be calculated using Equation (6.10). For the static model, assuming it is scheduled by EDF-VD and no service guarantee for LC tasks after mode-

Table 6.5: Relative improvement of MBEA+ on number of mode-switches

n_H	5	6	7	8	9
$U_A = 0.7, B(10, 0.4), RC = 3$	23.8%	32.3%	36.1%	48.8%	54.2%
$U_A = 0.8, B(10, 0.4), RC = 3$	43.0%	37.5%	44.2%	55.4%	27.9%
$U_A = 0.7, B(10, 0.6), RC = 3$	18.6%	26.3%	31.4%	34.6%	29.8%
$U_A = 0.8, B(10, 0.6), RC = 3$	17.0%	20.7%	27.2%	31.3%	32.5%
$U_A = 0.7, B(10, 0.4), RC = 5$	32.7%	38.0%	35.8%	37.9%	40.8%
$U_A = 0.8, B(10, 0.4), RC = 5$	31.7%	49.6%	46.8%	66.7%	42.9%
$U_A = 0.7, B(10, 0.6), RC = 5$	22.0%	23.4%	26.6%	25.8%	19.9%
$U_A = 0.8, B(10, 0.6), RC = 5$	16.3%	20.8%	24.7%	33.8%	20.6%

 Figure 6.15: $U_A = 0.7, RC = 3$ and $P(s_i) \sim B(10, 0.4)$

switch, the total system utilization is maximized when $\sum_{\tau_i \in \tau_H} C_i(L)/T_i = (1 - M)U_H$. Thus, the total system utilization for the static model can be represented as

$$SU(w) = w.U_L + w.(1 - U_L)(1 - U_H)/U_L + (1 - w)U_H$$

Here w denotes the weight for LC mode. Since there is no service guarantee for LC tasks, the utilization is maximized by postponing the mode-switch as much as possible, i.e., similar to setting $\beta^* = 1 - M$ in the dynamic model.

Figures 6.23 and 6.24 show the ratios⁶ between the total system utilization of dynamic and static models as a function of weight $w \in \{0.02, 0.04, \dots, 1.0\}$ when $U_L + U_H = 1.1$ or 1.3, respectively. As we can observe, when the weight for LC mode is relatively small, there is a clear performance gap

⁶Note that, if the ratio is greater than 1, it means the dynamic model has a larger total utilization than the static model.

Figure 6.16: $U_A = 0.7$, $RC = 5$ and $P(s_i) \sim B(10, 0.4)$

Figure 6.17: $U_A = 0.8$, $RC = 3$ and $P(s_i) \sim B(10, 0.4)$

between the two models. After the weight w exceeds a certain value, their performance overlaps. This indicates that when the weight w exceeds a certain value, the total system utilization is maximized when $\beta^* = 1 - M$. That is, if the weight for the LC mode is high enough and therefore we do not care about guaranteed service for LC tasks in the HC mode, then it is best to maximize the LC system service level for HC tasks (β^*).

Figure 6.18: $U_A = 0.8$, $RC = 5$ and $P(s_i) \sim B(10, 0.4)$

Figure 6.19: $U_A = 0.7$, $RC = 3$ and $P(s_i) \sim B(10, 0.6)$

6.8 Conclusions and Future Work

In this chapter, we proposed a dynamic LC budget allocation mechanism for HC tasks to overcome the limitations of static LC budget allocation. Unlike the static model, where the LC budget of each HC task is assumed to be equal to its WCET estimation for LC mode, the dynamic model determines it at runtime based on observed job execution times. Thus, the dynamic model is able to distribute the budgets more efficiently based on runtime information.

Figure 6.20: $U_A = 0.7$, $RC = 5$ and $P(s_i) \sim B(10, 0.6)$

Figure 6.21: $U_A = 0.8$, $RC = 3$ and $P(s_i) \sim B(10, 0.6)$

The dynamic model applies a novel mode-switch triggering scheme. The system switches its execution mode to HC only when the total LC budget allocation for all HC tasks combined is exceeded. As a result, the mode-switch is less likely to be triggered. In fact, we showed that this scheme is optimal in terms of being able to postpone mode-switches when compared to any strategy that uses a fixed budget allocation for each HC task.

We also proposed a mechanism that enables LC tasks to receive a minimum guaranteed budget

Figure 6.22: $U_A = 0.8$, $RC = 5$ and $P(s_i) \sim B(10, 0.6)$

Figure 6.23: Total system utilization when $U_L + U_H = 1.1$

allocation at all times, even in the HC mode. Then we presented metrics and explored the trade-off between the total LC budget allocation for HC tasks and the minimum guaranteed service for LC tasks.

A future research direction is to further explore the trade-off between budget allocation for HC tasks and service guarantee for LC tasks. In particular, techniques that can use the knowledge of task execu-

Figure 6.24: Total system utilization when $U_L + U_H = 1.3$

tion times (e.g., probabilistic WCET) to maximize the expected system utilization can be developed.

Another direction of research is improving the runtime strategy for budget allocation to HC tasks. In MEBA (and MEBA+) we use the maximum observed execution time among all jobs of a task to determine the budgets (i.e., e_i^m). However it may be possible to further postpone the mode-switch by using other parameters like the sum of executions of all jobs in the past.

Chapter 7

Conclusions and Future Work

In order to have confidence in the conclusions drawn from the schedulability analysis of a system abstracted by a workload model, the modeling process usually incorporates considerable amounts of pessimism. For example, the WCET estimation of code segments tends to involve very conservative assumptions about the system (cache flushing on preemption, no speculative executions, etc.) that are very unlikely to occur in practice. The WCET used in the modeling process is typically a safe upper bound that is significantly greater than the actual WCET. As a result, the platform resources that were provisioned based on the pessimistic WCET estimations are underutilized at runtime.

Mixed-Criticality (MC) scheduling theory seeks to deal with the above inefficiencies by modeling and analyzing the critical and less critical functionalities, separately. More pessimistic models are used to verify the temporal correctness of more critical functionalities that require a higher level of assurance. On the other hand, less critical functionalities are analyzed using less pessimistic models, because their temporary failure would not cause safety issues.

In this chapter, we first present a summary of the main contributions in this dissertation, and then discuss some possible future research directions.

7.1 Summary of Contributions

Vestal's workload model has abstracted some key properties of mixed-criticality systems. For example, it can guarantee the correctness of functionalities with different criticality levels to different degrees of confidence. More pessimistic WCET estimations are used for more critical functionalities whose correctness must be validated to a higher degree of confidence, while less critical functionalities are

verified less pessimistically. Because of this property, it has a higher resource utilization than traditional non-mixed-criticality models. As a result, a significant amount of scheduling policies and schedulability tests are proposed for this workload model.

Among the existing studies, a demand bound function (DBF) based analysis for EDF-VD* named GreedyTuning [24] has exhibited the best performance in terms of schedulability for a variety of synthetically generated mixed-criticality systems. However, it suffers from two drawbacks: 1) its schedulability performance drops significantly as the number of criticality levels increases, and 2) its performance in scheduling dual-criticality task systems with a larger percentage of high-criticality (HC) tasks is poor. One reason for these drawbacks is that it conservatively assumes maximum carry-over interference when the system behavior switches from a lower criticality level to the one being analyzed.

Based on the above observation, we derived an improved DBF based schedulability test as introduced in Chapter 4. This test collectively bounds the demand of the system in two consecutive execution modes, and uses maximum carry-over interference from previous execution modes if any. This improved the accuracy of the test significantly, with only a quadratic increase in the test runtimes. Further, we also developed a new virtual deadline assignment strategy for EDF-VD*, which effectively utilizes the new test to identify a candidate task for deadline reduction. Experiments show that our proposed test together with the new deadline assignment strategy significantly outperforms GreedyTuning [25], and its performance scales well with increasing number of criticality levels.

Although Vestal's workload model can improve resource utilization when compared to traditional non mixed-criticality models, it has a lot of unrealistic assumptions. These assumptions severely limit the applicability of studies based on this model. For example, many studies based on this model immediately suspend all the low-criticality (LC) tasks in the event that a single HC task executes beyond its WCET estimation for LC mode. This large-scale suspension is justified under the assumption that when a single HC task has an increased resource requirement, it is likely that all the other HC tasks in the system also have an increased resource requirement. However, this assumption is unrealistic, because it is not necessary that such a dependence may exist between all HC tasks. Hence, this collective suspension of LC tasks seems unreasonable and can lead to resource under-utilization.

To overcome the above limitation, and provide differentiated service to the LC tasks depending on the runtime behavior of specific HC tasks, we introduced the component-based mixed-criticality workload and execution model in Chapter 5. The proposed mechanism assumes all tasks are allocated to different components. Each component has a tolerance limit (TL) parameter to model the expected number of HC tasks that would require additional resources at the same time. As long as the number of HC tasks within

a component that exhibit HC behavior (i.e., require additional resources) does not exceed TL, tasks in other components are isolated from this component. Thus, LC tasks in other components continue to get full service even though some of the HC tasks in the system have already started consuming additional resources. The component boundaries are thus used in this model to contain the impact of HC tasks upto a certain extent. DBF based schedulability tests for both flat (non-hierarchical) as well as hierarchical scheduling frameworks are also derived for this model under the EDF-VD* scheduling policy. Since component-based system design is a popular paradigm in the safety-critical industry (e.g., AUTOSAR in automotive [9], ARINC-653 in avionics [8]), the proposed model is a natural fit in this paradigm and hence is practically relevant.

Another aspect of Vestal's workload model is that the WCET estimations of a task, particularly the LC mode estimations, are primarily used as budgets to control execution mode-switches. Thus, all previous works on MC scheduling share the same mode-switch condition, i.e., an HC task exceeds its WCET estimation for LC mode. It is important to note that these LC mode WCET estimations have no direct impact on the execution correctness of HC tasks; they are always required to receive as much budget as their HC mode WCET estimations. Since deriving these multiple estimations for a single task imposes a significant burden on the designer, it would be better if we could derive these LC mode budgets independent of the WCET estimations. Besides reducing the designer workload, such an approach could also improve performance and support for LC tasks, if for example the budgets are derived based on schedulability considerations.

Motivated by the above observations, we developed the dynamic MC task and scheduling model in Chapter 6. Basically, this model redefined the mode-switch condition based on budgets derived with schedulability considerations, so that mode-switches can either be postponed or avoided. In this model, each task has only one WCET estimation which is the HC mode estimation. Off-line, we determine a total LC budget allocation (single value) for all HC tasks combined based on schedulability requirements. At runtime, the system will penalize the LC tasks only when this total LC budget is violated. So as to minimize this penalty for LC tasks, we also developed a runtime budget allocation strategy called MEBA for HC tasks such that the mode-switches are postponed as much as possible. Finally, since tasks may not always consume their allocated budgets due to execution-time variations, we also extended this runtime strategy to reclaim unused budgets so that the mode-switches can be further postponed (MEBA+).

Thus, this dissertation has successfully addressed the important problems related to mixed-criticality systems identified in Chapter 1. Nevertheless, several other related problems remain open, and we

present some of them in the following section.

7.2 Future Research Directions

Most existing studies in MC scheduling try to answer one or both of the following two questions:

- Q(1): When should the system penalize LC tasks? Almost all the existing studies (except [64]) assume that some LC tasks must be penalized once an HC task τ_i executes beyond its LC mode budget $C_i(L)$. Such approaches completely ignore the degree to which τ_i violates $C_i(L)$. For example, different strategies could be applied when τ_i executes for $C_i(L) + 0.01(C_i(H) - C_i(L))$ time units or $C_i(H)$ time units.
- Q(2): How should the LC tasks be penalized so as to guarantee the correctness of HC tasks? There are various answers to Q(2). For example, some of them (see [16,23–25,32,33]) completely abandon all the LC tasks, and others (see [19–22,58,64]) offer degraded service to LC tasks.

Alternate solutions for Q(1): The proposed dynamic model in Chapter 6 provides a different answer to Q(1) by making use of the maximum execution time among all jobs of τ_i in the current busy interval (i.e., e_i^m). However, its effectiveness would be impacted if the system has a very long busy interval or if there is a lot of variation in the job execution times of a task. *Hence, one possible future research direction is to find better answers to Q(1) by making use of historical information such as exact execution times of previous jobs.* Based on this information, it might be feasible to distribute the budgets more efficiently among tasks than what was possible using MEBA+. This could further postpone or avoid mode-switches even during long busy intervals.

Type of Overload: In the existing MC scheduling theory, there is no consideration for whether the system experiences a transient or a constant overload. If the system experiences a transient overload, i.e., the HC tasks require additional resources only for a short duration, then it would recover to normal execution mode very quickly and only a few LC jobs would be impacted. In this case, most LC jobs can still meet deadlines. However, if a system experiences a constant overload (e.g., because of changes to the operating environment), then the LC tasks are likely to be penalized for a very long time duration. However, this does not meet our expectation where most LC jobs can meet their deadlines.

Therefore, another possible future research direction is to explore strategies that can distinguish a transient and a constant overload. In the former case, we may still apply existing techniques in MC

scheduling theory. However, in the latter case, we should consider different strategies such as speeding up the processor frequency, so that most LC jobs can still meet deadlines. To develop such strategies, it would be essential to first develop overload models so that the different conditions can be identified.

Dual Priority MC Scheduling: Dual priority (DP) scheduling first proposed by Burns and Wellings [68] introduces minimum dynamic-priority behavior when compared to traditional fixed-priority scheduling by promoting the priority of each job only once. With this simple technique, DP scheduling achieves much better schedulability than fixed-priority scheduling. Under DP scheduling, each task has an initial priority that undergoes priority promotion at a fixed time instant (called promotion point) relative to the release time of each job. Suppose we use DP scheduling for mixed-criticality systems, in particular we use priority promotions for HC tasks. Consider an HC task τ_i that has an initial priority P and promotion point P_i at which the priority of the task is promoted to priority level P' with $P' < P$ (lower priority value implies higher fixed-priority). A small P_i value enables the scheduler to favor τ_i that has low initial priority, so that it can meet its deadline even if its need for additional resources is revealed much after its release.

Given a priority assignment and a promotion point assignment, the simplest analysis for DP scheduling of mixed-criticality system is to check:

1. Whether all tasks can meet their deadlines while none of the HC tasks require any additional resources, and
2. Whether all HC tasks can meet their deadlines if they execute upto their HC mode WCET estimations.

Thus, another possible research direction is to explore the trade-off between the promotion points and priorities, and apply DP in the scheduling of mixed-criticality systems.

LC Support on Multiprocessors: Most existing strategies for supporting LC tasks (e.g., the dynamic model in Chapter 6) are designed for uniprocessor platforms. If we want to extend such techniques to multiprocessor platforms, we need to deal with the usual problems such as partitioning of tasks to different processors or deriving appropriate schedulability tests for multiprocessor platforms. Alternately, such platforms also offer the possibility of migrating LC tasks from processors that are experiencing overload to the ones that are executing in the normal mode. Thus, another possible future research

direction is to extend the strategies that are designed for uniprocessor platforms to multiprocessor platforms, taking into consideration such additional capabilities for LC support.

References

- [1] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, “Deadline scheduling for real-time systems: Edf and related algorithms.” Springer Science & Business Media, 2012.
- [2] A. K.-L. Mok, “Fundamental design problems of distributed systems for the hard-real-time environment,” Ph.D. dissertation, Massachusetts Institute of Technology, 1983.
- [3] C. Liu and J. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [4] A. K. Mok and D. Chen, “A multiframe model for real-time tasks,” *IEEE transactions on Software Engineering*, vol. 23, no. 10, pp. 635–645, 1997.
- [5] S. K. Baruah, “Feasibility analysis of recurring branching tasks,” in *Proceedings of Euromicro Workshop on Real-Time Systems*, 1998, pp. 138–145.
- [6] S. K. Baruah, “A general model for recurring real-time tasks,” in *Proceedings of IEEE Real-Time Systems Symposium*, 1998, pp. 114–122.
- [7] P. J. Prisaznuk, “Integrated modular avionics,” in *Proceedings of the IEEE National Aerospace and Electronics Conference*, 1992, pp. 39–45.
- [8] W. River, “Arinc 653—an avionics standard for safe, partitioned systems,” in *IEEE Seminar*, 2008.
- [9] A. Consortium *et al.*, “Automotive open system architecture,” <http://www.autosar.org/>, 2009.
- [10] R. Bell, “Introduction to iec 61508,” in *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software - Volume 55*, ser. SCS '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 3–12. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1151816.1151817>
- [11] L. A. Johnson, “Do-178b, software considerations in airborne systems and equipment certification,” *Crosstalk*, October, 1998.
- [12] V. Hilderaman and T. Baghi, *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)*. Avionics Communications, 2007.

REFERENCES

- [13] P. Sinha, "Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives," *Reliability Engineering & System Safety*, vol. 96, no. 10, pp. 1349–1359, 2011.
- [14] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2007, pp. 239–243.
- [15] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2017.
- [16] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling Real-Time Mixed-Criticality Jobs," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140–1152, 2012.
- [17] A. Burns and S. Baruah, "Towards a More Practical Model for Mixed-Criticality Systems," in *Proceedings of Workshop on Mixed-Criticality Systems*, 2013.
- [18] "Mixed-Criticality & Real-Time in Automotive," 2017, dagstuhl Seminar 17131 ? Mixed Criticality on Multicore / Manycore Platforms. [Online]. Available: <http://materials.dagstuhl.de/files/17/17131/17131.DirkZiegenbein.Slides.pdf>
- [19] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptations for mixed-criticality systems," in *In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.
- [20] H. Su, N. Guan, and D. Zhu, "Service guarantee exploration for mixed-criticality systems," in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014, pp. 1–10.
- [21] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2016, pp. 35–46.
- [22] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15. New York, NY, USA: ACM, 2015, pp. 237–246. [Online]. Available: <http://doi.acm.org/10.1145/2834848.2834850>
- [23] A. Easwaran, "Demand-based Scheduling of Mixed-Criticality Sporadic Tasks on One Processor," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2013, pp. 78–87.
- [24] P. Ekberg and W. Yi, "Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2012, pp. 135–144.

REFERENCES

- [25] P. Ekberg and W. Yi, “Bounding and shaping the demand of generalized mixed-criticality sporadic task systems,” *Real-Time Systems*, vol. 50, no. 1, pp. 48–86, 2014.
- [26] A. Hergenhan and G. Heiser, “Operating systems technology for converged ecus,” in *6th Emb. Security in Cars Conf.(escar)*. Hamburg, Germany: ISITS, 2008.
- [27] S. Baruah, A. Mok, and L. Rosier, “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 1990, pp. 182–190.
- [28] H. Li and S. Baruah, “An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2010, pp. 183–192.
- [29] N. Guan, P. Ekberg, M. Stigge, and W. Yi, “Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2011, pp. 13–23.
- [30] S. Baruah, A. Burns, and R. Davis, “Response-Time Analysis for Mixed Criticality Systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2011, pp. 34–43.
- [31] S. Baruah, V. Bonifaci, G. D’Angelo, H. Li, and A. Marchetti-Spaccamela, “The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems,” in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2012, pp. 145–154.
- [32] S. Baruah, V. Bonifaci, G. D’Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, “Mixed-criticality Scheduling of Sporadic Task Systems,” in *Proceedings of the European Symposium on Algorithms*, 2011, pp. 555–566.
- [33] S. Baruah, V. Bonifaci, G. D’angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, “Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems,” *Journal of the ACM (JACM)*, vol. 62, no. 2, p. 14, 2015.
- [34] S. Baruah, H. Li, and L. Stougie, “Towards the Design of Certifiable Mixed-criticality Systems,” in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 13–22.
- [35] S. Baruah, A. Easwaran, and Z. Guo, “Mixed-criticality scheduling to minimize makespan,” in *LIPICs-Leibniz International Proceedings in Informatics*, vol. 65. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- [36] N. Audsley, “Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times,” in *The University of York Technical Report*, 1991.
- [37] F. Dorin, P. Richard, M. Richard, and J. Goossens, “Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities,” *Real-Time Systems*, vol. 46, no. 3, pp. 305–331, 2010.

REFERENCES

- [38] S. Baruah and A. Burns, “Implementing Mixed Criticality Systems in Ada,” in *Proceedings of the Ada-Europe International Conference on Reliable Software Technologies*, 2011, pp. 174–188.
- [39] T. Fleming, “Extending mixed criticality scheduling,” Ph.D. dissertation, University of York, 2013.
- [40] Q. Zhao, Z. Gu, and H. Zeng, “Pt-amc: Integrating preemption thresholds into mixed-criticality scheduling,” in *Design, Automation & Test in Europe Conference & Exhibition*, 2013, pp. 141–146.
- [41] A. Burns and R. I. Davis, “Adaptive mixed criticality scheduling with deferred preemption,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2014, pp. 21–30.
- [42] S. Baruah, A. Burns, and R. I. Davis, “An extended fixed priority scheme for mixed criticality systems,” *Proceedings of Workshop on Real-Time Mixed Criticality Systems*, pp. 18–24, 2013.
- [43] S. Baruah and S. Vestal, “Schedulability Analysis of Sporadic Tasks with Multiple Criticality Specifications,” in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2008, pp. 147–155.
- [44] T. Park and S. Kim, “Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems,” in *Proceedings of the ninth ACM international conference on Embedded software*, 2011, pp. 253–262.
- [45] A. Masrur, D. Müller, and M. Werner, “Bi-level deadline scaling for admission control in mixed-criticality systems,” in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015, pp. 100–109.
- [46] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, “Interference constraint grapha new specification for mixed-criticality systems,” in *IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, pp. 1–8.
- [47] J. Ren and L. T. X. Phan, “Mixed-criticality scheduling on multiprocessors using task grouping,” in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2015, pp. 25–34.
- [48] B. Hu, K. Huang, P. Huang, L. Thiele, and A. Knoll, “On-the-fly fast overrun budgeting for mixed-criticality systems,” in *Proceedings of the IEEE & ACM International Conference on Embedded Software*, 2016, pp. 1–10.
- [49] F. Santy, L. George, P. Thierry, and J. Goossens, “Relaxing Mixed-Criticality Scheduling Strictness for Task Sets Scheduled with FP,” in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2012, pp. 155–165.
- [50] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar, “Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems,” in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, 2013, pp. 183–192.

REFERENCES

- [51] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *Proceedings of the Euromicro Conference on Real-Time Systems*, 2015, pp. 259–268.
- [52] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013, pp. 147–152.
- [53] H. Su, P. Deng, D. Zhu, and Q. Zhu, "Fixed-priority dual-rate mixed-criticality systems: Schedulability analysis and performance optimization," in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2016, pp. 59–68.
- [54] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll, "Adaptive runtime shaping for mixed-criticality systems," in *Proceedings of the IEEE & ACM International Conference on Embedded Software*, 2015, pp. 11–20.
- [55] M. Neukirchner, K. Lampka, S. Quinton, and R. Ernst, "Multi-mode monitoring for mixed-criticality real-time systems," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2013, p. 34.
- [56] P. Holman and J. H. Anderson, "Group-based pfair scheduling," *Real-Time Systems*, vol. 32, no. 1, pp. 125–168, 2006.
- [57] X. Gu and A. Easwaran, "Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 1, pp. 24:1–24:24, Nov. 2017.
- [58] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *Proceedings of the Euromicro Conference on Real-Time Systems*, July 2015, pp. 13–24.
- [59] S. Baruah and G. Fohler, "Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2011, pp. 3–12.
- [60] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality system," in *Proceedings of Workshop on Mixed-Criticality Systems*, 2013.
- [61] "Deos: A Time & Space Partitioned DO-178 Level A Certifiable RTOS," http://www.ddci.com/products_deos.php.
- [62] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of IEEE Real-Time Systems Symposium*, 2003, pp. 2–13.
- [63] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework using EDP Resource Models," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2007, pp. 129–138.

REFERENCES

- [64] X. Gu and A. Easwaran, “Dynamic budget management with service guarantees for mixed-criticality systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2016, pp. 47–56.
- [65] R. I. Davis, K. W. Tindell, and A. Burns, “Scheduling slack time in fixed priority pre-emptive systems,” in *Proceedings of IEEE Real-Time Systems Symposium*, 1993, pp. 222–231.
- [66] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.
- [67] T. Fleming and A. Burns, “Incorporating the notion of importance into mixed criticality systems,” in *Proceedings of Workshop on Mixed Criticality Systems*, 2014, p. 33.
- [68] A. Burns and A. Wellings, “Dual Priority Assignment: A Practical Method of Increasing Processor Utilisation ,” in *Proc. of the Fifth Euromicro Workshop on Real-time Systems*, 1993.