Fall 2018

Mixed-criticality real-time task scheduling with graceful degradation

Samsil Arefin

MIXED-CRITICALITY REAL-TIME TASK SCHEDULING WITH GRACEFUL DEGRADATION

by

SAMSIL AREFIN

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2018

Approved by:

Zhishan Guo, Advisor
Donald C. Wunsch
Patrick Taylor

ABSTRACT

The mixed-criticality real-time systems implement functionalities of different degrees of importance (or criticalities) upon a shared platform. In traditional mixed-criticality systems, under a HI mode switch, no guaranteed service is provided to LO-criticality tasks. After a mode switch, only HI-criticality tasks are considered for execution while no guarantee is made to the LO-criticality tasks. However, with careful optimistic design, a certain degree of service guarantee can be provided to LO-criticality tasks upon a mode switch. This concept is broadly known as graceful degradation. Guaranteed graceful degradation provides a better quality of service as well as it utilizes the system resource more efficiently. In this thesis, we study two efficient techniques of graceful degradation.

First, we study a mixed-criticality scheduling technique where graceful degradation is provided in the form of minimum cumulative completion rates. We present two easy-to-implement admission-control algorithms to determine which LO-criticality jobs to complete in HI mode. The scheduling is done by following deadline virtualization, and two heuristics are shown for virtual deadline settings. We further study the schedulability analysis and the backward mode switch conditions, which are proposed and proved in (Guo et al., 2018).

Next, we present a probabilistic scheduling technique for mixed-criticality tasks on multiprocessor systems where a system-wide permitted failure probability is known. The schedulability conditions are derived along with the processor allocation scheme. The work is extended from (Guo *et al.*, 2015), where the probabilistic model is first introduced for independent task scheduling on a uniprocessor platform. We further consider the failure dependency between tasks while scheduling on multiprocessor platforms.

We provide related theoretical analysis to show the correctness of our work. To show the effectiveness of our proposed techniques, we conduct a detailed experimental evaluation under different circumstances.

ACKNOWLEDGMENTS

I would like to express my gratitude to everyone for whom it became possible to accomplish my research work successfully. First of all, I would like to thank Dr. Zhishan Guo, my advisor, whose immense knowledge and constant support helped me to achieve my goal. It was a great opportunity for me to work in his research group, which helped me to learn a lot about the real-time systems research area, and to collaborate with numerous talented researchers. Without his proper guidance, it wouldn't be possible for me to overcome all the hurdles I faced during my master's program and in this research work.

Furthermore, I would like to thank my committee members, Dr. Donald C. Wunsch and Dr. Patrick Taylor, for agreeing to be the part of my thesis committee and for giving their valuable time to review this work. I am grateful to Missouri S&T, and more specifically the computer science department, for providing me the opportunity and sufficient facilities to perform my research work. Finally, I want to thank my parents and my wife, Farzana Shawarna, for their constant support and love.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

NOMENCLATURE

AMC	Adaptive Mixed-Criticality
BF	Best-Fit
BFD	Best-Fit Decreasing
DBF	Demand Bount Function
EDF	Earliest Deadline First
EDF-GVD	Earliest Deadline First - Graceful Virtual Deadline
EDF-VD	Earliest Deadline First with Virtual Deadline
EVT	Extreme Value Theory
FF	First-Fit
FFD	First-Fit Decreasing
GEDF-VD	Global Earliest Deadline First with Virtual Deadline
LF	Largest-First
MC	Mixed-Criticality
MIT	Minimum Inter-arrival Time
pET	probabilistic Execution Time
pMC	probabilistic Mixed-Criticality
pMCMP	probabilistic Mixed-Criticality on MultiProcessor

pWCET	probabilistic Wost-Case Execution Time
QoS	Quality of Service
RAD	Reasonable Allocation Decreasing
WCET	Wost-Case Execution Time
WF	Worst-Fit
WFD	Worst-Fit Decreasing

1. INTRODUCTION

Real-Time Systems refers to the infrastructure where the temporal correctness is as much important as the logical correctness. While the logical correctness of the system ensures that the correct results are produced, the temporal correctness focuses on the completion of a specific task at the right time (e.g., within deadlines). In simple words, we can describe the simplest form of a real-time system as follows — if there is a set of tasks to be completed with given deadlines by using a specific amount of resources, the goal is to complete all tasks on time (e.g., within their corresponding deadlines). To accomplish this goal properly, we need an efficient scheduling algorithm. With the advancement of technology as well as the real-time systems itself, various types of such system emerge, hence raising numerous new challenges.. Based on different system infrastructure, requirements, and constraints, there exist different real-time systems and different scheduling problems. For example, based on the nature of the tasks, there can be periodic or sporadic task systems. Based on the criticality level, there can be single-criticality or mixed-criticality systems. Depending on the platforms where the tasks will be scheduled, there can be uniprocessor or multiprocessor scheduling problems. The goal of a scheduling algorithm is to accomplish the scheduling challenge (i.e., fulfilling both logical and temporal correctness). The real-time scheduling theory allows us to study such scheduling algorithms or create a new one for a new challenge, and also to validate the algorithms by deriving schedulability tests to guarantee temporal correctness.

In this thesis, we focus on the scheduling problem of mixed-criticality real-time task systems. Along with the mixed-criticality task scheduling, our main goal is to provide graceful degradation (enhanced quality of service) to low critical tasks. We study two

different scheduling problems of such systems which are some of the emerging problems in the area. This section provides a brief introduction to the different problems which are studied in this thesis, followed by the contribution and organization.

1.1. MIXED CRITICALITY SYSTEMS

In a real-world system, where multiple applications are considered with a variety of task sets, not every application have the same importance level as others. Either based on fulfilling the mission of the application or to enhance the quality of service (QoS), success of some of these applications are more *critical* than others. For example, a navigation system in an airplane is obviously more critical than a real-time entertainment system. Here the navigation system can be seen as a mission-critical task, failing to accomplish such tasks on time can have a drastic effect and may create risks for hundreds of lives. However, the entertainment system can be considered as a quality-critical task, which is good to have but occasionally failing to complete such tasks wouldn't harm a lot. Due to the size, weight, and power considerations, there is a trend in combining such mixed-critical applications of different degree of importance with varying specifications upon a shared platform (Burns and Davis, 2017c) and such systems are called *Mixed-Criticality (MC) systems*.

Due to the emerging importance of MC systems, government and industrial organizations (including AFRL, NSF, NSA, NASA, etc.) have put much effort into it in recent years. For example, the software standards in the European automotive industry (AUTOSAR) and in the avionics domain (ARINC) already recognize the importance of MC on their platforms (Guo and Baruah, 2018) while the US Air Force has acknowledged that the MC architectures are needed where *safety* and *security* are designed for certification (Barhorst *et al.*). Not only the government and industrial organizations but also the academia realize the strategic significance of the study of such systems. A numerous amount of research has been done (Burns and Davis, 2017a) since the pioneering work

by Vestal (of Honeywell Aerospace) in 2007 (Vestal, 2007). In Table 1.1, some current industrial system standards and their corresponding failure conditions are shown which are originally presented in (Guo and Baruah, 2018).

Table 1.1. Safety levels defined by different industrial standards

Standards	ISO26262 (ASIL)	IEC62304 (Class)	IEC61508 (SIL)	EN50128 (SSIL)	DO-178B/C (DAL)	Failure Condition
Safety Levels	A	A	1	0/1	A	Catastrophic
	B	-	2	2	B	Hazardous
	C	B	2	2	C	Major
	D	C	3	3	D	Minor
	-	-	4	4	E	No Effect

Most of the early research on scheduling MC task system is based on the Vestal model (Vestal, 2007), which defines the correctness as follows: *all deadlines will be met under normal circumstances, while if some more important tasks overrun, a mode switch is triggered and only* HI-*critical deadline will be guaranteed to met.* In traditional MC system, different worst-case execution time (WCET) are specified for each HI-criticality task for different mode. For example, in a two-criticality-level system, each task can be of either higher (HI) or lower (LO) criticality. Two WCET estimations, a LO-WCET and a HI-WCET, are specified for each HI-criticality task (usually the HI-WCET is several orders of magnitude larger than the LO-WCET). However, only one WCET is specified for each LO-criticality task. Whenever there is a single failure by any HI-criticality task, i.e., the task does not finish before the deadline, a *system-wide mode switch* will be triggered. Upon a mode switch, all HI-criticality tasks are allocated by using there HI-WCET estimation while all the LO-crtiticality tasks are dropped.

1.2. GRACEFUL DEGRADATION IN MIXED-CRITICALITY SYSTEMS

Most of the prior research on MC schedulability (Please refer to (Burns and Davis, 2017b) for an up-to-date thorough review) is based on the Vestal model (Vestal, 2007), where upon a mode switch, all LO-criticality tasks are dropped. Although this system-mode based MC model is quite successful to solve some of the core challenges, however, it fails to identify a major challenge which is pointed by systems engineers and researchers (Burns, 2017) :

> LO-criticality functionalities are *not* non-critical — they should be guaranteed with some *degree of service*, regardless of HI-criticality tasks' behaviors.

GRACEFUL DEGRADATION: The term *graceful degradation* of any system refers to the ability to maintain limited functionality even under an adverse situation, when a large portion of the system is destroyed or does not work properly. Scheduling algorithm with graceful degradation allows LO-criticality tasks to receive a certain amount of service, instead of fully being abandoned, even after a mode switch. Variety of applications can produce better result upon integrating such algorithms in the system. For example, Baruah et al. (Branicky *et al.*, 2002) show that a control signal processing in the networked control system can still perform stably while skipping a limited number of computation tasks within a certain. Similarly, this technique can be applied in a continuous closed-loop system where completing a minimal fraction of some computations (tasks) on time can only cause an optimal disturbance rejection performance (Majumdar *et al.*, 2011).

In the real-time systems community, the study of MC scheduling with graceful degradation become popular in recent years (Baruah, 2015; Baruah *et al.*, 2016; Gettings *et al.*, 2015; Guo *et al.*, 2015; Liu *et al.*, 2016; Saha *et al.*, 2015). However, at least one of the following issues remain unsolved in most of the existing works:

1. In utilization based schedulers, LO-criticality utilization upon a mode switch is reduced. As a result, either the execution time is reduced or the period is increased. A reduced execution time for each task may lead to unfinished execution (malfunction)as LO-WCET is already optimistically estimated (with tight margin). On the other hand, a longer period may result in the loss of timeliness which results in a degraded quality of service as well.

2. Some schedulers maintain asymptotic bounds to provide graceful degradation but it doesn't provide any fine granularity. For example, to maintain a 20% graceful degradation in asymptotic bound, both 1-out-of-4 and $10k$-out-of-$40k$ are considered as correct. However, it is obvious that the latter one provides a lower quality of service.

By considering the above situation, it is a demand of time to provide a better scheduling algorithm which identifies both of the above scenarios.

1.3. PROBABILISTIC MC TASK SCHEDULING

As described in the Section 1.1, upon a mode switch in a traditional MC system, it is assumed that, after a HI mode switch, all HI-criticality tasks exceed their LO-WCET budget *simultaneously*. As a result, all LO-criticality tasks are dropped to make place for the execution of HI-criticality tasks. However, the assumption that all HI-criticality tasks exceeding their HI-WCET simultaneously is rather pessimistic. It may result in unwanted wastage of system resource as well as the degradation of quality as all LO-criticality tasks are not further considered for execution. However, if the probability of all the HI-criticality tasks exceeding their HI-WCET was considered, there might have been better schedulability and QoS for the task system. The scenario can be understood more clearly by the following example by (Guo *et al.*, 2015):

Example 1.3.1. *Consider a MC system with two independent* HI*-criticality tasks* τ_1 *and* τ_2*. Each task has a* LO*-criticality utilization* u_{LO} *and a* HI*-criticality utilization* u_{HI} *where* $u_{\text{LO}} \leq u_{\text{HI}}$*. The utilization values for both tasks are as follows:* $\tau_1 = \{0.4, 0.6\}$ *and* $\tau_2 = \{0.3, 0.5\}$*. It is clear that this task system cannot be scheduled on a preemptive uniprocessor platform as the system utilization in* HI*-criticality mode is 1.1 while the processor capacity is 1.*

In traditional real-time scheduling and MC scheduling, the absolute certainty of correctness is required, i.e., any single failure is considered as an overall system failure. In such cases, the task set in example 1 can never be scheduled. However, let's consider the following constraints for the task set: (i) The probability of any HI*-criticality job exceeding its* LO*-WCET budget is estimated as* 10^{-4} *per hour. (ii) The permitted failure probability for the system is* 10^{-6} *per hour. and (iii) The tasks are independent, i.e., the failure of one task doesn't affect another task.*

Based on the above assumption, the probability of jobs from both tasks exceeding their LO*-WCETs is* $10^{-4} \times 10^{-4} = 10^{-8}$ *per hour which is less than the permitted failure probability* 10^{-6} *per hour. As a result, the system is probabilistically feasible for schedule as the total remaining utilization at worst case will be* $\max(0.4 + 0.3, 0.4 + 0.5, 0.6 + 0.3) = 0.9 \leq 1$.

From the above example, it is clear that the task set with specific requirements can be schedulable using a probabilistic scheduling algorithm without dropping any LO-criticality task (i.e., without compromising QoS) while the traditional algorithms are unable to schedule them. Upon successful scheduling, the probabilistic scheduler provides a specific type of graceful degradation to the system. Here the LO-criticality tasks are guaranteed to execute until the permitted failure probability is not violated. As a result, this probabilistic scheduler can provide better QoS to the system. Hence, the probabilistic scheduling becomes one of the key research areas in both real-time scheduling and MC scheduling.

1.4. CONTRIBUTION AND ORGANIZATION

As discussed above, significant work on MC scheduling has been done in the real-time systems community. However, there exists the need for efficient scheduling techniques with graceful degradation in several demanding yet unsolved areas. Our work primarily focuses on these areas while studying the MC scheduling with graceful degradation and probabilistic schedulability in multiprocessor platforms. We provide efficient algorithms along with experimental results. The main contributions of this thesis are:

- In this work, we present an MC scheduling algorithm based on r-out-of-n graceful degradation. We present detailed experimental results based on the schedulability analysis proposed in (Guo et al., 2018).
- This thesis studies multiprocessor probabilistic scheduling of MC systems and provides necessary algorithms and schedulability test along with detailed experimental results based on randomly generated task sets.

ORGANIZATION: The rest of the thesis is organized as follows: Section 2 consists of the detailed literature review of previous works. In Section 3, the system models used in this thesis is described. Section 4 contains a noble MC scheduling algorithm with graceful degradation while Section 5 presents the multiprocessor probabilistic MC task scheduling algorithm and the corresponding results. Finally, Section 6 consists of a summary of the thesis.

2. LITERATURE REVIEW

In the previous section, we have introduced the concepts of MC scheduling, MC scheduling with graceful degradation, and probabilistic scheduling, which are directly related to the contents of this thesis. In this section, we will briefly go through the literature reviews over the state-of-the-art works done on those areas.

2.1. MC REAL-TIME TASK SCHEDULING

Since Vestal's first proposal (Vestal, 2007) on the concept of MC workload model, numerous amount of work has been done on MC task scheduling (see (Burns and Davis, 2017a) for an up-to-date review). However, determining the schedulability of an MC instance is proven NP-hard in strong sense even under the *uniprocessor platforms*. (Baruah *et al.*, 2012a). As a result, different scheduling techniques such as fixed priority scheduling (Baruah *et al.*, 2010, 2011a,c; Guan *et al.*, 2011; Li and Baruah, 2010) as well as dynamic priority scheduling (such as *E*arliest *D*eadline *F*irst (EDF)) (Baruah *et al.*, 2015; Chen *et al.*, 2014; Easwaran, 2013; Li, 2013; Masrur *et al.*, 2015) are studied over the time to provide efficient approximate algorithms. It has been proven that, in uniprocessor platforms, an MC task scheduling algorithm can achieve optimal speedup factor[1] of at most $4/3$ (Baruah *et al.*, 2011c).

Due to the importance and increasing demand for multi-core platforms, works have been coduced on MC scheduling algorithms for such platforms as well (Baruah, 2004; Gratia *et al.*, 2015a,b; Li and Baruah, 2012). While many works propose MC scheduling by extending existing standard multi-processor scheduling algorithms (Awan *et al.*, 2017; Bletsas and Petters, 2012; Guo, 2016; Niz *et al.*, 2009; Rodriguez *et al.*, 2013; Xu and

[1]For a scheduler $\mathcal{S}$, a speedup factor $\mathcal{V}$ ($\mathcal{V} \geq 1$) (also known as resource augmentation factor), means that any task set that is schedulable on a platform of speed-1 core will be schedulable by $\mathcal{S}$ on a platform where each core is of speed $\mathcal{V}$.

Burns, 2015) different techniques are introduced in different works such as fluid-based MC model (Lee *et al.*, 2014), global fixed priority scheduling algorithms (Baruah, 2004; Pathan, 2012), use of hierarchy of servers (Gratia *et al.*, 2015a,b), globally scheduled fixed-priority systems (Pathan, 2012), and a semi-partition based scheme (Awan *et al.*, 2017). However, the performance of multiprocessor scheduling algorithms is not as good as the uniprocessor ones. GEDF-VD (Global Earliest Deadline First with Virtual Deadline) (Li and Baruah, 2012) provides the highest speedup factor of $(\sqrt{5} + 1)$.

2.2. MC SCHEDULING WITH GRACEFUL DEGRADATION

While most of the existing MC scheduling algorithms provide guarantees for the completion of HI-criticality tasks quite efficiently, LO-criticality tasks don't get any service guarantee upon a mode switch. To identify this issue, Baruah et al. (S. Baruah and A. Burns, 2014) presented an alternative model which allow LO tasks's execution after a criticality mode switch by assigning a lower priority level to them. However, in this model, all LO-criticality tasks can still miss their deadlines as no guarantee is provided.

Santy et al. (Santy *et al.*, 2012) proposed another algorithm where LO-criticality jobs get some service as long as the execution of HI-criticality jobs is not hindered. In (Jan *et al.*, 2013; Su and Zhu, 2013), elastic task model (also known as task stretching) is used where the LO-priority tasks receive dynamically enlarged periods and deadlines in higher modes. Fleming and Burnds first introduced the notion of 'importance' (Fleming and Burns, 2014), which provides more control over the degradation of service in more realistic system models.

Apart from the best-efforts algorithms mentioned above, several works have been done to provide som guarantee to LO-criticality tasks under HI mode. Baruah et al. (Baruah *et al.*, 2016) used fluid based techniques to provide a degraded (but non-zero) level of service to LO-criticality tasks under all non-erroneous behavior of the system. Liu et al. (Liu *et al.*, 2016) proposed an utilization-based schedulability test under EDF-VD (Earliest Deadline

First with Virtual Deadline) scheduling. In (Guo *et al.*, 2015), Guo et al. proposed a unique system model and corresponding schedulability analysis by incorporating failure probability inoformation into the mixed criticality task model.

Although the notion of graceful degradation is gaining more interest in recent years, the concept of graceful degradation can be traced back to the 1990's when Hamdaoui and Ramanathan (Hamdaoui and Ramanathan, 1995) propose (m, k)-firm model for streams. According to this model, out of consecutive k tasks, at least m deadlines must be met. A weakly-hard task model is introduced by Bernat et al. (Bernat *et al.*, 2001) which can tolerate a predefined degree of missed deadlines. In control system, an expected asymptotic success rate is introduced by Saha et al. (Saha *et al.*, 2015) for task scheduling. Adaptive Mixed Criticality (AMC) (Baruah *et al.*, 2011b) is used in an MC scheduling technique proposed by Getting et al. (Gettings *et al.*, 2015) where they incorporate weakly hard constraints with the graceful degradation. Delaying mode-switch is another approach used by Gu and Easwaran (Gu and Easwaran, 2016) where LO-criticality budgets for individual HI-criticality applications are determined dynamically during runtime. Among other recent works, Chwa et al.(Chwa *et al.*, 2018) proposed a new single-criticality cyber-physical system task model, where they maintain the stability of the system by exploring the trade-off between enlarging periods and consecutive task drops to guarantee schedulability.

2.3. PROBABILISTIC SCHEDULABILITY

In order to overcome the disadvantages of discreet MC task models, several probabilistic models have been proposed over time. Edgar and Burns (Edgar and Burns, 2001) first introduced the concept of *probabilistic confidence* to the real-time task and the system model. They have used estimated probabilistic WCETs (pWCETs) from test data for individual tasks and provided a suitable lower bound for the overall confidence level of a system. Since then, several works have been done with the focus in better WCET estimations and a predicted probability of exceeding such WCET along with the use of extreme value theory

(EVT) (Cucu-Grosjean *et al.*, 2012; Griffin and Burns, 2010; Hansen *et al.*, 2009). Other significant works consist probabilistic WCET estimations with preemptions, (Davis *et al.*, 2013), pWCET estimation (Hardy and Puaut, 2013; Slijepcevic *et al.*, 2013) in the presence of permanent faults and disabling of hardware elements, and probabilistic Execution Time (pET) estimation (David and Puaut, 2004) based upon a tree-based technique. The pWCET estimation is calculated based on the probability of a task exceeding a specific value, while the pET is the probability of a job's execution time being equal to a particular value.

Among the work regarding pWCET and pET estimation, Tia et al. (Tia *et al.*, 1995) provide two methods for probabilistic schedulability guarantees by focusing on the unbalanced heavy loaded system. Probabilistic schedulability analysis for earliest deadline first ((Zhu *et al.*, 2002) and fixed priority policy (Gardner and Liu, 1999) is deducted based on the initial work of Lehoczky (Lehoczky, 1996). A generic analysis for probabilistic systems with pWCET estimations for tasks is presented in (Díaz *et al.*, 2002). While most of the works are based on the WCET estimation, (Abeni and Buttazzo, 1999) and (Maxim and Cucu-Grosjean, 2013) focus on providing statistical guarantee upon the minimum inter-arrival time (MIT) estimation. Among the works which are based on pETs (instead of pWCETs), (Hansen *et al.*, 2002) focused on the limited priority level case (quantized EDF), and (Manolache *et al.*, 2004) presented an associated schedulability analysis on multiprocessors.

Guo et al. (Guo *et al.*, 2015) first proposed a unique probabilistic system model for MC real-time task model. They have introduced a failure probability for each HI task, a system-wide permitted failure probability, and provided schedulability analysis for the EDF-based scheduling algorithm. Our work is motivated by this work and we further extend this work to support multiprocessor platforms.

3. SYSTEM MODEL

In this section, first, we present the traditional MC model used by most of the state-of-the-art works and its corresponding system behavior and MC-correctness. Then, we demonstrate each system model used in this thesis, which are created by modification (mainly addition) of the traditional model. We also define some notations which are used throughout the paper.

In this thesis, we restrict our attention to MC sporadic task model. In such MC model, a task set τ consists of n finite amount of tasks $\tau_1, \tau_2, \ldots, \tau_n$. Every task $\tau_i \in \tau$ may produce infinite number of MC jobs. While the first job of each task can be released in the same time or different time, the analysis of different release times can be translated into a single release time analysis by finding the first time instance where the jobs of all the tasks are released. Without loss of generality, the release time for the first job of each task is assumed to be 0, i.e., all tasks in τ are released at time 0. Due to simplicity and ease of understanding, we also restrict our attention to dual-criticality task systems only, where criticality levels and execution modes are restricted to LO and HI.

3.1. TRADITIONAL MC SYSTEM MODEL

In traditional dual-criticality sporadic task system, each task $\tau_i \in \tau$ is characterized by 5-tuples:

$$\{C_i^{\text{LO}}, C_i^{\text{HI}}, T_i, D_i, \chi_i\} \tag{3.1}$$

Here, $C_i^{\text{LO}}, C_i^{\text{HI}} \in \mathbb{R}_+$ are the two WCET estimations for LO-criticality mode and HI-criticality mode respectively. T_i represents the period of a task, which is the minimum inter-arrival time between any two consecutive job releases. The deadline of the task is denoted by D_i while $\chi_i \in \{\text{LO}, \text{HI}\}$ represents the criticality level of each task. Note that, for implicit-deadline

task system period and deadline is same, i.e., $T_i = D_i$, while for constrained-deadline task system $T_i > D_i$. Usually, for LO-criticality tasks the value of C_i^{LO} and C_i^{HI} are same while for HI-criticality tasks, C_i^{HI} is greater than C_i^{LO}. So in general, we can assume that, $0 < C_i^{\text{LO}} \leq C_i^{\text{HI}} \leq T_i$.

The utilizations of the task set τ in LO-criticality and HI-criticality mode is calculated as follows:

$$\forall \tau_i \in \tau, u_i^{\text{LO}} = \frac{C_i^{\text{LO}}}{T_i};$$

$$\forall \tau_i \in \tau, u_i^{\text{HI}} = \frac{C_i^{\text{HI}}}{T_i}.$$

The total utilization for each mode of operation is calculated as follows:

- The total utilization for all LO-criticality tasks in LO- and HI-modes respectively are, $U_{\text{LO}}^{\text{LO}} = \sum_{\forall \tau_i \in \tau_{\text{LO}}} u_i^{\text{LO}}$, $U_{\text{LO}}^{\text{HI}} = \sum_{\forall \tau_i \in \tau_{\text{HI}}} u_i^{\text{LO}}$.
- Similarly, for all HI-criticality tasks, the utilization for HI-and LO-criticality tasks are: $U_{\text{HI}}^{\text{LO}} = \sum_{\forall \tau_i \in \tau_{\text{LO}}} u_i^{\text{HI}}$, $U_{\text{HI}}^{\text{HI}} = \sum_{\forall \tau_i \in \tau_{\text{HI}}} u_i^{\text{HI}}$.

In general, the utilizations for LO-criticality tasks in both HI- and LO-criticality modes are same, i.e., $U_{\text{LO}}^{\text{LO}} = U_{\text{LO}}^{\text{HI}}$.

SYSTEM BEHAVIOR: The traditional mixed criticality model has the following semantics.

- If every job is completed upon executing no more than the C^{LO} of the corresponding task, then we call it a LO-criticality behavior.
- if one or more HI-criticality jobs complete upon executing more than C^{LO} but no more than C^{HI}, then the system behavior is categorized as HI-criticality.
- All other behaviors are erroneous.

CORRECTNESS CRITERIA: An MC scheduling is correct if both the following properties are satisfied:

- During all LO-criticality behaviors of the system, each job receives an execution of up to its LO-criticality WCET between its release time and deadline (such that all jobs can be completed before their deadlines).

- During all HI-criticality behavior of the system, all HI-criticality jobs receive enough execution (up to its HI-criticality WCET) between their release time and deadline such that all such jobs can be completed before their deadline. In the meanwhile, all LO-criticality jobs are dropped and are not considered for further execution.

3.2. MC GRACEFUL DEGRADATION MODEL

While the traditional MC task model doesn't provide any service to LO-criticality tasks in HI-criticality mode, in this thesis, we study a noble approach of graceful degradation to MC task sets. The system model discussed in this section is first introduced in our previous work (Guo et al., 2018). In addition to the existing MC task model, for each LO-criticality task τ_i, a minimum cumulative admission rate r_i is introduced. Also, we provide separate model for task τ_i based on its criticality level $\chi_i \in \{\text{LO}, \text{HI}\}$. Each LO-criticality task τ_i can be characterized as:

$$\tau_i = \{C_i, T_i, D_i, r_i, \chi_i\}, \ \forall \tau_i \in \tau_{\text{LO}},$$

while each HI-criticality task τ_i can be characterized as:

$$\tau_i = \left\{C_i^{\text{LO}}, C_i^{\text{HI}}, T_i, D_i, \chi_i\right\}, \ \forall \tau_i \in \tau_{\text{HI}}.$$

where, LO-criticality task set $\tau_{\text{LO}} = \{\tau_i | \chi_i = \text{LO}\}$ and the HI-criticality task set $\tau_{\text{HI}} = \{\tau_i | \chi_i = \text{HI}\}$.

While the model for HI-criticality tasks and all other parameters of LO-criticality tasks are characterized similarly to traditional MC task model as described before, the value of r_i, which is introduced in LO-criticality tasks model, denotes that, at HI mode, the

completion rate of that task must be at least r_i. That is, for the first N jobs released by τ_i after the mode switch (to the HI mode), at least $\lceil r_i \cdot N \rceil$ number of jobs should be completed. Furthermore, each LO-criticality task τ_i has only a single WCET estimation C_i alongside required completion rate r_i while each HI-criticality task τ_i has two WCET estimations like the traditional MC model.

As described in (Guo et al., 2018), the above system model is a *generalization of many existing models* as well as *bridges mixed-criticality* and *non-mixed-criticality scheduling*. As the value of r_i may vary for each LO-criticality task, when they are all zero, our problem becomes a traditional sporadic MC task scheduling problem. On the other hand, an all 1 values turn the system into a non-mixed-criticality scheduling problem as no jobs can be dropped.

CORRECTNESS CRITERIA: The correctness criteria are defined in (Guo et al., 2018) for the above MC task system. An MC scheduling is correct if both the following properties are satisfied:

- During all LO-criticality behaviors of the system, each job receives an execution of up to its LO-criticality WCET between its release time and deadline (such that all jobs can be completed before their deadlines).
- During all HI-criticality behavior of the system, all HI-criticality jobs receive enough execution (up to its HI-criticality WCET) between their release time and deadline such that all such jobs can be completed before their deadline. In the meanwhile, LO-criticality jobs will be executed following a minimum cumulative admission rate r_i; i.e., starting from the last HI mode switch point, out of any $N \in \mathbb{Z}^+$ consecutive jobs released by any LO-criticality task τ_i, $\lceil r_i \cdot N \rceil$ jobs *are guaranteed to receive full execution (between their release time and the deadlines)*.

3.3. SYSTEM MODEL WITH PERMITTED FAILURE PROBABILITY

For the probabilistic schedulability, we follow the system model proposed by Guo et al (Guo *et al.*, 2015). According to their work, in an MC task model with failure probability, HI-criticality tasks are represented by:

$$\tau_i = (C_i^{\text{LO}}, C_i^{\text{HI}}, T_i, D_i, f_i, \chi_i)$$

LO-criticality tasks continue to be represented with three parameters as before.

Note that, A *failure probability* parameter f_i is added to HI-criticality task τ_i, which represents the probability of the actual execution requirement of *any* job of the task exceeding C_i^{LO} (but still below C_i^{HI}) in one hour. Furthermore, an *allowed system failure probability* F_S is specified. It describes the permitted probability of the system during one hour of execution. F_S may be very close to zero (e.g., 10^{-12} for some safety-critical avionics functionalities).

In this thesis, we consider implicit-deadline task models for probabilistic schedulability, i.e., $T_i = D_i$. Furthermore, unlike the work in (Guo *et al.*, 2015), we consider scheduling such task system in multiprocessor platforms.

CORRECTNESS CRITERIA: The correctness criteria for the probabilistic schedulability is inherited from the work in (Guo *et al.*, 2015). An MC scheduling is correct if both the following properties are satisfied:

- If the probability of any task missing its deadline is no greater than F_S, the task system is strongly probabilistic schedulable,
- If the probability of any HI-criticality task missing its deadline is no greater than F_S, and no deadline is missed when all jobs finish upon execution of their LO-WCETs, If it returns weakly probabilistic schedulable

3.4. THESIS STATEMENT

New methods with an optimistic analysis of MC real-time task scheduling can be used to provide a higher degree of services to LO-criticality to enhance the overall quality of service of MC systems. Efficient MC scheduling technique can be implemented to provide a reduced yet non-zero guaranteed service to LO tasks under HI mode by maintaining a cumulative completion rate. Furthermore, accommodating the probabilistic scheduling in MC scheduling, especially on multiprocessor platforms, can increase the schedulability of the task system, hence improving the overall quality of service.

4. MIXED-CRITICALITY TASK SCHEDULING WITH GRACEFUL DEGRADATION

In Section 1, we presented the concept of graceful degradation and discussed the drawbacks of existing MC scheduling algorithms which provide graceful degradation to LO-criticality tasks upon a mode switch. In this section, we study a noble approach which guarantees a specific amount of service to LO-tasks even after a HI-mode switch.

In Section 3, we discuss the system model used for the algorithm we present in this section. To reiterate briefly, we study a new MC system model as proposed in (Guo et al., 2018), which redefines the HI-criticality mode and provides service guarantee to LO-criticality tasks. On top of that, we study the following topics in this section:

- We study the minimum cumulative admission rate r_i which ensures that upon a system-wide mode switch, for any $N \in \mathbb{Z}^+$, at least $\lceil r_i \cdot N \rceil$ out of N new consecutive job releases (by task τ_i) will receive full execution.

- An easy-to-implement admission control procedure to LO-criticality tasks upon mode switch is proposed to guarantee the rate r_i for any task τ_i. Moreover, if a rate r_i can be expressed in the fraction form $r_i = m_i/k_i$, then our procedure further guarantees that after mode switch occurs, at least m_i jobs will receive full execution out of **any** k_i consecutive jobs of a LO-criticality task τ_i.

- We adapt the earliest-deadline-first (EDF) with virtual deadlines scheduler and proposed a pseudo-polynomial time schedulability test for the MC system based on demand-bound function (DBF) analysis.

- We include a backward mode switch (from HI mode back to LO mode) mechanism, then prove the maximum length of the period (upper bounds) to bring the system back to LO mode safely, with our scheduler, under different scenarios of execution patterns to HI-criticality tasks.

- To demonstrate the effectiveness of our algorithm, we verify the theoretical results by conducting the experiments via simulation.

4.1. ADMISSION CONTROL OF LO-CRITICALITY TASKS IN HI MODE

In our MC graceful degradation model, every LO-criticality task has a minimum cumulative admission rate r_i which determines the HI mode system behavior. In HI-criticality mode, HI-criticality tasks need more execution time budget (i.e., HI-WCET). As a result, due to lack of resource in HI mode, it is obvious that not every of LO-criticality tasks can be executed in every scenario. Therefore, to maintain any $r_i < 1$, we must drop some of the LO-criticality jobs. However, the dropping of LO-criticality jobs needs to respect the minimum cumulative admission rate r_i; i.e., for any task τ_i, if $\lceil r_i \cdot N \rceil$ out of the first N consecutive jobs after the mode switch are executed ($\forall N \in \mathbb{Z}^+$), then r_i is satisfied. To accomplish this goal we need to design an admission control protocol, where two different cases are considered: (A) when r_i is a rational number, and (B) when it is not. (refer to (Guo et al., 2018))

(A). When r_i is a rational number for task τ_i, it can be represented in a fractional form (m_i/k_i), where m_i and k_i are co-prime integer values and $gcd(m_i, k_i) = 1$. In this case, we can generate a repeated pattern μ_i of size k, such that m_i out of k_i jobs are selected for execution. This admission control pattern μ_i is generated off-line and used dynamically to decide which jobs to admit and which jobs to drop under the HI mode.

If the cumulative admission rate without considering the current job is less than r_i, then it should be admitted for execution, otherwise, this job can be dropped. Following this observation, we design Algorithm 1 for the admission control protocol. To make it simple it is enough to check the condition ($a < b \times r$) at every position b, where a is the successful number of admission so far. For any $i > k$ the admission of a job should be controlled by the value $\mu[i \bmod k]$.

Algorithm 1: Static Admission Control

```
Data: r_i values for any LO-criticality task τ_i
Result: vector μ_i: μ_i ∈ {0,1}^{k_i} is the admission control pattern for each LO-criticality
        task τ_i
foreach τ_i ∈ τ_LO do
    a = 0;                                  // counts successful admission
    k_i = Get_Min_Int_Denominator(r_i);
    for b = 1 to k_i do
        if (a < b × r_i) then
            μ_i[b] = 1 ;                    // admit
            a = a + 1;
        end
        else
            μ_i[b] = 0 ;                    // drop
        end
    end
end
return μ_i;
```

The output of Algorithm 1, μ_i, is a binary vector of length k_i, representing the admission control pattern of the input LO-criticality task τ_i. In μ, value 1 at any position i denotes that the job of that position is allowed to execute while value 0 is denoted for dropped jobs. For example, the pattern {10100} denotes that job 1 and 3 are admitted, while job 2, 4, and 5 are dropped. To ensure a uniform distribution, the algorithm maintains that at each position, the successful admission ratio is at least r_i.

(B). If r_i is not a rational number, any repeated pattern (μ_i) with a limited length cannot be guaranteed like the previous case. In this scenario, we make the admission-control decision dynamically on-line, which is presented in Algorithm 2. Here we control the admission of LO-criticality jobs similar to Algorithm 1, but instead of prior calculation of pattern μ, we make the admission decision during run-time.

Algorithm 2: Dynamic Admission Control

```
Data: (r_i) values for all LO-criticality tasks
foreach τ_i ∈ τ_LO do
    a_i = 0;                                   // Number of executed jobs
end
while true do
    if j_b ∈ τ_i released then
        // j_b is b^th job of τ_i
        if (a_i < b × r_i) then
            // schedule j_b using EDF
            a_i = a_i + 1;
        end
        else
            // The job j_b is dropped
        end
    end
end
```

We now show how the two admission control schemes work with an example of several tasks.

Example 4.1.1. *Consider a task set τ where the requirements of minimum cumulative admission rates are given in Table 4.1, where S_i denotes the maximum number of consecutive drops to a certain task (which is a function of r_i).*

Table 4.1. Sample completion rates with associated admission patterns and maximum job acceptance separation

Task ID	r_i	Pattern μ_i	S_i
τ_1	0.4	$\{10100\}^\infty$	2
τ_2	0.625	$\{11011010\}^\infty$	1
τ_3	$1/\sqrt{2}$	$11101101110\cdots$	1

First, consider τ_1 to determine the μ. If we consider dropping the job at position 1 then the condition $(a < b \cdot r)$ is satisfied. So the first job will be allowed to execute. For the second job whereby considering allowing the job, our admission control condition doesn't meet. Similarly by checking up to position k the values of μ becomes 10100*. All jobs of task τ_1 will be determined by this pattern. So the admission control pattern of task τ_1 will be* $(10100)^{\infty}$*. Note that, its cumulative admission rate (the proportion of admitted jobs over all released jobs) pattern is* $1, 1/2, 2/3, 1/2, 2/5, 1/2, 3/7, 1/2, 4/9, 2/5, \cdots$*, which never drops below $r_i = 0.4$. The admission control pattern for τ_2 can be determined similarly and the pattern is* $(11011010)^{\infty}$.

However, For τ_3, the value of r_i is an irrational number. So, we calculate the admission control dynamically by checking similar condition $(a < b \cdot r)$. Hence the admission control for τ_3 is done in a non-repetitive manner; i.e., $11101101110 \cdots$.

Remark 1. This dynamic approach of Algorithm 2 can also be used for specific case even when the r is a rational number with a large denominator. In Algorithm 1, we compute an admission control pattern μ_i prior to run-time. If we consider m_i/k_i as the standard form of every minimum degradation rate r_i, then it requires $\sum_{i \in n} k_i$ amount of space to store all the patterns. When r_i has a large denominator, the dynamic approach of Algorithm 2 can be used to save the extra space for those cases even r_i is rational. Consider the case when a set consisting one (or more) value/s of r_i is given in a way such that the value of k_i is too large, then it may not be feasible to store this large pattern/s in the system and we can use the dynamic approach to schedule the LO-criticality jobs in HI-criticality mode. The threshold value k for applying Algorithm 1 or 2 in admission control can vary from system to system and hence should be decided by the system engineer.

Remark 2. One important aspect of our admission control protocol is that we minimize the maximum number of consecutive drops $S(r_i)$ while maintaining the minimum cumulative admission rate r_i. $S(r_i)$ can also be viewed as the maximum number of consecutive zeros in

μ_i. The following equation reveals the relationship between $S(r_i)$ and r_i:

$$S(r_i) = \left\lceil \frac{1}{r_i} \right\rceil - 1. \tag{4.1}$$

Remark 3. Upon a mode switch (to HI mode), for any $N \in \mathbb{Z}^+$ consecutive job releases of any LO-criticality task τ_i, the admitted number of jobs equals to $\lceil r_i \cdot N \rceil$.

4.2. SCHEDULER AND SCHEDULABILITY ANALYSIS

In this section, we consider MC constrained-deadline sporadic task system on a preemptive uniprocessor while maintaining the minimum cumulative admission rate r_i. for LO-criticality tasks One of the most efficient approaches for scheduling such tasks is to use *virtual deadlines* where the HI-criticality tasks get their deadline reduced(if necessary) during execution in LO-criticality mode. We use a similar technique to define an algorithm EDF-GVD (Graceful-Virtual-Deadline)(refer to (Guo et al., 2018)), which is derived from the concept of EDF-VD (Baruah *et al.*, 2012b).

In this section, we first describe our EDF-GVD algorithm and propose two virtual deadline setting mechanisms which are followed by the proof of correctness and necessary schedulability analysis for different modes.

4.2.1. Algorithm EDF-GVD. Let $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ denote the targeted MC constrained-deadline sporadic task set, to be scheduled on a uniprocessor platform. Prior to runtime, Algorithm EDF-GVD performs a schedulability test (Section 4.2.2) based on demand bound function to determine whether the task set τ can be scheduled or not. If τ is deemed schedulable, then we calculate the virtual deadline $D_i^v \leq D_i$ for each HI-criticality task, which is later used to schedule the corresponding tasks using EDF.

VIRTUAL DEADLINE SETTINGS: We set the virtual deadlines by following similar techniques like other EDF-VD familiy algorithm. We first calculate the virtual deadlines D_i^v for HI-criticality tasks we first use a simple yet efficient heuristic such that the

per-mode utilizations are the same for each task, i.e.,

$$D_i = \frac{C_i^{\text{LO}}}{C_i^{\text{HI}}} D_i. \tag{4.2}$$

Since the above heuristic may not provide an suitable result for every scenario, we use an alternative approach for setting virtual deadlines:

$$D_i = q \cdot D_i, \ s.t. \ \forall i \ \chi_i = \text{HI}. \tag{4.3}$$

Here, $q \in (0, 1]$ is the common virtual deadline scaling factor for the whole set, to be determined using Algorithm 3.

Algorithm 3: Virtual Deadline Setting

```
Data: Task set τ, Accuracy ϵ
Δ = 0.5 //step size;
q = 0.5 //virtual deadline shrinking parameter;
while Δ ≥ ϵ do
    Δ = Δ/2;
    foreach τ_i ∈ τ_HI do
        D_i = q · D_i;
    end
    C_A = Check(Condition (A) in Section 4.2.2);
    C_B = Check(Condition (B) in Section 4.2.2);
    if C_A = true && C_B = true then
        Return q;
    end
    else if C_A = true && C_B = false then
        q = q − Δ;
    end
    else if C_A = false && C_B = true then
        q = q + Δ;
    end
    else
        Return failure; //no q can be found
    end
    Return -1; //still possible with a smaller ϵ
end
```

To find a feasible q to any level of preciseness, we perform a binary search over the range $(0, 1]$. In short, Algorithm 3 performs the following two tasks:

1. Binary search is performed over the range $(0, 1]$ up to a desired degree of accuracy, and find the smallest value of q for which all the HI-criticality tasks with virtual deadline $q \cdot D_i$ along with all the LO-criticality tasks with their original deadline are schedulable. (Satisfying 4.2.2-condition A)

2. For the value of q determined above, check if all the HI-criticality tasks with their original deadlines along with a certain amount of LO-criticality jobs maintaining the minimum degradation rate r_i are schedulable. (Satisfying 4.2.2-condition B)

TIME COMPLEXITY: Algorithm 3 runs in pseudo-polynomial time (to a desired degree of accuracy, say 2^{-10}) as the main steps depends on the schedulability test to be mentioned in Section 4.2.2, which takes pseudo-polynomial time for any q.

RUN-TIME BEHAVIOR: After computing the feasible virtual deadlines D_i, run-time scheduling for all tasks is done in an EDF manner. Specifically, under the LO mode, jobs of each HI-criticality task τ_i are assigned a virtual deadline of D_i after their releases, while the relative deadline of a LO-criticality job remains D_i. If some HI-criticality job does not signal its completion upon receiving a cumulative execution time of its LO-criticality WCET, i.e., during the HI-criticality behavior of the system, the tasks are scheduled as follows:

- If $\chi = \text{HI}$, then this job is assigned a virtual deadline equal to its original deadline D_i.

- If $\chi = \text{LO}$, then the execution of this job is determined based on the admission control protocol described in Section 4.1. If the job is selected for execution then it is assigned a virtual deadline equal to its original deadline D_i.

Example 4.2.1. *Table 4.2 shows a MC system consisting of three tasks.*

The task set is EDF-schedulable under LO *mode as the system density (i.e.,* $\sum_i C_i/D_i = 1/6 + 1/3 + 2/4$*) is* 1.

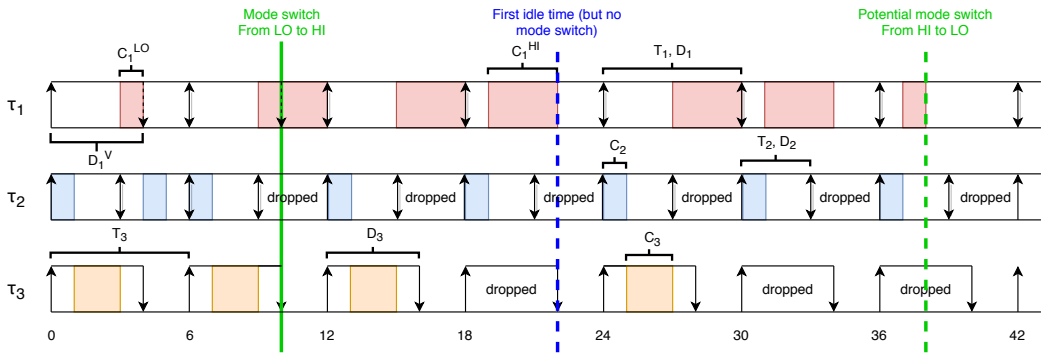


Figure 4.1. EDF-GVD scheduling of the task set provided in Example IV.1

Table 4.2. An MC set with minimum degradation execution rates

Task ID	C_i (C_i^{LO})	C_i^{HI}	T_i	D_i	χ_i	r_i
τ_1	1	3	6	6	HI	-
τ_2	1	-	3	3	LO	0.5
τ_3	2	-	6	4	LO	0.4

However, Under the traditional MC sporadic task model, after the mode switch all the LO*-criticality tasks (τ_2 and τ_3) are dropped and are not considered for further execution as the overall system utilization (i.e., $\sum_i C_i/T_i = 3/6 + 1/3 + 2/6 = 7/6$) becomes greater than* 1*. However, the task system is schedulable using EDF-GVD as shown in Figure 4.1.*

During LO*-criticality mode, the* HI*-criticality task τ_1 is scheduled using virtual deadline $D_1 = 4$. At time $t = 10$, the second job of τ_1 couldn't finish its execution withing its estimated* LO*-WCET budged (i.e., $C_1^{\text{LO}} = 1$). As a result, a system wide mode switch to* HI*-criticality mode occurs. In this scenario, the algorithm drops all the pending jobs of all* LO*-criticality tasks (e.g., (4th)* LO*-criticality job of τ_2 is dropped in this case). In* HI*mode, the* HI*-crticality task τ_1 is scheduled using EDF with $C_1^{\text{HI}} = 3$ and deadline $D_1 = 6$, and the* LO*-criticality jobs are scheduled based on the admission control procedure described in Section 4.1 (see 'dropped' jobs in the figure). The system is ready for a potential mode switch to* LO*-mode again when:*

- *The processor is idle,*

- *The last jobs of all* HI*-criticality tasks are completed within its* C^{LO} *amount of time (denoted by dashed green lines), and*

- *The last job of any* HI*-criticality task didn't execute more than its* C^{LO}

If the last job of any HI*-criticality task executes more than its* C^{LO}*, there cannot be any backward mode switch even if the system is idle (denoted by dashed blue lines)*

4.2.2. DBF Based Schedulability Analysis. In this section, we study the schedulability test to check the correctness of both LO and HI mode based on the virtual deadline setting from Section 4.2.1. To prove the correctness, we use the *demand bound function* (DBF), which has been demonstrated to be a successful approach to analyze the schedulability of both ordinary (Baruah *et al.*, 1990) and mixed-criticality (Ekberg and Yi, 2014) real-time systems.

Definition 1. ($\text{dbf}(\tau_i, \ell)$ function.) The function $\text{dbf}(\tau_i, \ell)$ denotes the maximum execution demand of task τ_i during *any* time interval of length ℓ, where the *demand* is calculated by the cumulative execution requirement of all jobs of τ_i that have both release times and deadlines in that interval.

Baruah et al. (Baruah *et al.*, 1990) proposed and proved the following dbf-based schedulability test for non-mixed criticality tasks:

Lemma 1. *A constrained-deadline task set* τ *can be successfully scheduled by EDF on a uniprocessor, if*

$$\forall \ell \geq 0, \sum_{\tau_i \in \tau} \text{dbf}(\tau_i, \ell) \leq \ell.$$

Ekberg et al. (Ekberg and Yi, 2014) propose an efficient method to analyze the DBF for MC systems. In this thesis, we use the following four DBF functions:

- $\text{dbf}^{\text{LO}}_{\text{LO}}(\tau_i, \ell)$—the demand bound of a LO-criticality task τ_i in LO mode for any time interval of length ℓ.

- $\mathrm{dbf}^{\mathrm{LO}}_{\mathrm{HI}}(\tau_i, \ell)$—the demand bound of a HI-criticality task τ_i in LO mode for any time interval of length ℓ.
- $\mathrm{dbf}^{\mathrm{HI}}_{\mathrm{LO}}(\tau_i, \ell)$—the demand bound of a LO-criticality task τ_i in HI mode for any time interval of length ℓ.
- $\mathrm{dbf}^{\mathrm{HI}}_{\mathrm{HI}}(\tau_i, \ell)$—the demand bound of a HI-criticality task τ_i in HI mode for any time interval of length ℓ.

Definition 2. (Carry-Over Job.) *A carry-over job is a job that is released before the mode switch and has a deadline after the mode switch.*

The carry-over jobs do not affect the calculation of $\mathrm{dbf}^{\mathrm{LO}}_{\mathrm{LO}}(\tau_i, \ell)$ and $\mathrm{dbf}^{\mathrm{LO}}_{\mathrm{HI}}(\tau_i, \ell)$. Furthermore, in the algorithm EDF-GVD, carry over jobs in the calcuation of $\mathrm{dbf}^{\mathrm{HI}}_{\mathrm{LO}}(\tau_i, \ell)$ is not considered either as it drops all the such jobs of LO-criticality tasks. Hence, we only need to calculate the carry over jobs while calculating $\mathrm{dbf}^{\mathrm{HI}}_{\mathrm{HI}}(\tau_i, \ell)$. While calculating this value, we can consider carry-over jobs as a special job that releases at the time of mode switch and has an execution of the remaining execution requirement that has not completed before the mode switch. As a result, we can use the following schedulability test which is directly extended from Lemma 1.

Lemma 2. *A mixed-criticality system τ can be successfully scheduled by EDF on a uniprocessor if both of the following conditions hold.*

Condition (A):

$$\forall \ell \geq 0, \sum_{\tau_i \in \tau_{\mathrm{LO}}} \mathrm{dbf}^{\mathrm{LO}}_{\mathrm{LO}}(\tau_i, \ell) + \sum_{\tau_i \in \tau_{\mathrm{HI}}} \mathrm{dbf}^{\mathrm{LO}}_{\mathrm{HI}}(\tau_i, \ell) \leq \ell.$$

Condition (B):

$$\forall \ell \geq 0, \sum_{\tau_i \in \tau_{\mathrm{LO}}} \mathrm{dbf}^{\mathrm{HI}}_{\mathrm{LO}}(\tau_i, \ell) + \sum_{\tau_i \in \tau_{\mathrm{HI}}} \mathrm{dbf}^{\mathrm{HI}}_{\mathrm{HI}}(\tau_i, \ell) \leq \ell.$$

Please refer to (Guo et al., 2018) for the detail proof of Lemma 2.

TIME COMPLEXITY OF SCHEDULABILITY TEST: In Conditions (A) and (B), it is stated that "$\forall \ell$" needs to be assessed, which implies an unbounded number of dbf computations of different values of ℓ. In follwoing Lemmas, it is shown that only a limited number of ℓ values are enough for the calculation as shown in (Guo et al., 2018), which yields a schedulability test with pseudo-polynomial time complexity.

Lemma 3. *Let c be a constant such that $c < 1$ and $\sum_{\tau_i \in \tau_{\text{LO}}}(C_i/T_i) + \sum_{\tau_i \in \tau_{\text{HI}}}(C_i^{\text{LO}}/T_i) \leq c$. Then, Condition (A) is true for $\forall \ell \geq 0$, if it is true for all ℓ such that*

$$\ell < \frac{c}{1-c} \cdot \max\{\max_{\tau_i \in \tau_{\text{LO}}}\{T_i - D_i\}, \max_{\tau_i \in \tau_{\text{HI}}}\{T_i - D_i^v\}\}.$$

Lemma 4. *Let c_1 and c_2 be two constants such that $\sum_{\tau_i \in \tau_{\text{LO}}}(r_i \cdot C_i/T_i) \leq c_1$, $\sum_{\tau_i \in \tau_{\text{HI}}}(C_i^{\text{HI}}/T_i) \leq c_2$, and $c_1 + c_2 < 1$. Then, Condition (B) is true for $\forall \ell \geq 0$, if it is true for all ℓ such that*

$$\ell < \frac{c_1 \cdot \max_{\tau_i \in \tau_{\text{LO}} \wedge r_i > 0}\{T_i - D_i + \frac{T_i}{r_i}\} + c_2 \cdot \max_{\tau_i \in \tau_{\text{HI}}}\{T_i - D_i + D_i^v\}}{1 - c_1 - c_2}.$$

The proofs of Lemma 1 and 2 are given in (Guo et al., 2018).

4.2.3. Mode Switch in Both Directions. One of the major advantages of our work is that we can easily determine the possibility of backward mode switch from HI to LO mode. In traditional Vestal model (Vestal, 2007), it is extremely unlikely that there will be a mode switch during run-time (Baruah, 2018). As a result, the backward mode switch calculation is often skipped in such models. However, in MC scheduling with graceful degradation, the LO-WCET estimation is done optimistically and are designed to be violated from time to time, thus it is important to determine the mode switches to both directions. Hence, in our work, we take HI-to-LO mode switch into consideration. As mentioned in (Guo et al., 2018), we propose a more realistic backward mode switch behavior:

Whenever some HI-criticality task overruns, the system will switch to HI mode immediately. The system is again switched back to LO-criticality mode when: (i) the processor *idles* in HI-criticality mode, and (ii) the last instance of each HI-criticality task is finished within its LO-WCET, where all further LO-criticality releases are accepted and all deadlines are met.

In the following Lemma, we provide a mathematical bound for the backward mode switch instance. We suppose

$$\sum_{\tau_i \in \tau_{\text{HI}}} \frac{C_i^{\text{LO}}}{T_i} + \sum_{\tau_i \in \tau_{\text{LO}}} \frac{r_i \cdot C_i}{T_i} \leq \sum_{\tau_i \in \tau_{\text{HI}}} \frac{C_i^{\text{HI}}}{T_i} + \sum_{\tau_i \in \tau_{\text{LO}}} \frac{r_i \cdot C_i}{T_i} < 1,$$

which is, in fact, required for our schedulability test.

Lemma 5. *Let t_d denote the absolute deadline of the latest* HI*-criticality job that overruns its* LO*-WCET in this* HI *mode. Then, there must be an idle time instant at or before $t_d + L_1$, where*

$$L_1 = \frac{\sum_{\tau_i \in \tau_{\text{HI}}} (2C_i^{\text{LO}}) + \sum_{\tau_i \in \tau_{\text{LO}}} (C_i + 2r_i \cdot C_i)}{1 - \sum_{\tau_i \in \tau_{\text{HI}}} (C_i^{\text{LO}}/T_i) - \sum_{\tau_i \in \tau_{\text{LO}}} (r_i \cdot C_i/T_i)}.$$

Please refer to (Guo et al., 2018) for the detail proof of Lemma 5.

4.3. EXPERIMENTS

To evaluate the performance of EDF-GVD, we compare the results to the existing state-of-the-art algorithms. In this section, we present the detail experimental results. To demonstrate the efficiency of our proposed algorithm, several simulations are conducted considering a series of different scenarios. For comparison, we have used EDF-VD (Baruah *et al.*, 2012b), EDF-VD with (m, k) guarantee (under density based analysis), and EDF-VD with shrunk C^{LO} (Liu *et al.*, 2016) (which provides weaker guarantees). The similarity, speedup-optimality for uniprocessor MC task scheduling were the major factors behind choosing these specific EDF-VD family algorithms.

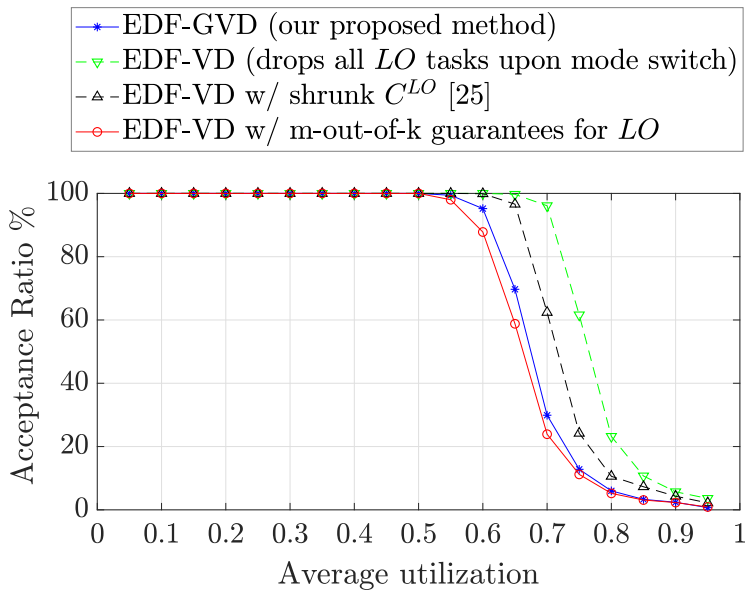
4.3.1. Workload Generation. We use a similar approach to (Ekberg and Yi, 2014) for generating the MC workload. We generate random MC sporadic tasks based on following parameters (here U stands for uniform distribution):

- $P_{\text{HI}} = 0.5$: An individual task's probability of being HI-criticality. The criticality level χ_i is decided based on this parameter.
- $C_i^{\text{LO}} \sim U[1, 10]$: The values of C_i^{LO} (and C_i) are uniformly generated from this range.
- $R_{\text{HI}} = 4$: It denotes the ratio of C_i^{HI} to C_i^{LO}; if $\chi_i = \text{HI}$, then the value of C_i^{HI} is uniformly generated from the range $[C_i^{\text{LO}}, R_{\text{HI}} \cdot C_i^{\text{LO}}]$.
- $T_{max} = 200$: The value of period T_i is uniformly generated from the range $[C_i(\chi_i), T_{max}]$.
- $minDR \sim U[0.1, 0.9]$: This value is used to generate the deadline after generating $minDR$ uniformly from given range, the deadline is calculated by following $D_i = \alpha \cdot T_i$; where α is uniformly generated from the range $[minDR, 1]$.

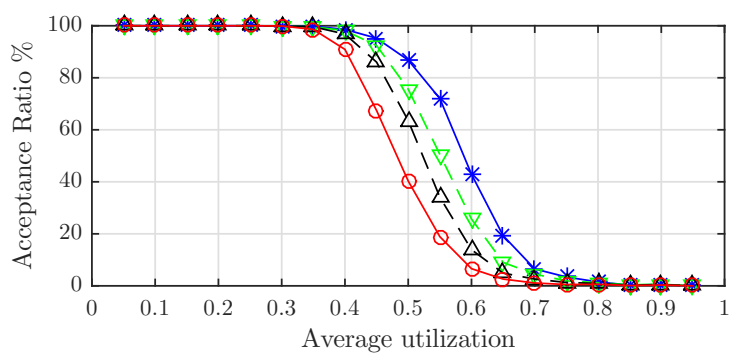
We use a fixed *target average utilization* U^* to generate the tasks of each task set τ. Once a task is generated, we take the average of U_{LO} and U_{HI} to calculate the *average utilization* $U(\tau)$. The task set is generated in a ways such that the average utilization is near the value of U^*. To accomplish this we generate the average utilization from the range $[U^*, U^*]$, where $U^* = U^* - 0.005$ and $U^* = U^* + 0.005$.

We keep generating task for a specific task set τ as long as $U(\tau) < U^*$. The whole task set is discarded if at any point the $U(\tau)$ becomes larger than U^*. A generated task set is considered for further processing if the value of $U(\tau)$ is within the range $[U^*, U^*]$. However, if all the tasks of τ have same criticality level, $U_{\text{LO}}(\tau) > 0.99$, or $U_{\text{HI}}(\tau) > 0.99$, the task set is discarded immediately.

For the three algorithms used in comparison, we calculate the utilization of a task by using C_i/D_i instead of C_i/T_i as they only considered implicit-deadline MC task models. Also, we set $C_i^{\text{HI}} = r_i \cdot C_i^{\text{LO}}$ for EDF-VD with shrunk C_i^{LO}, as the requirement in HI mode for LO-criticality task is $C_i^{\text{LO}} \geq C_i^{\text{HI}}$. We present our experimental results in Figure 4.2.



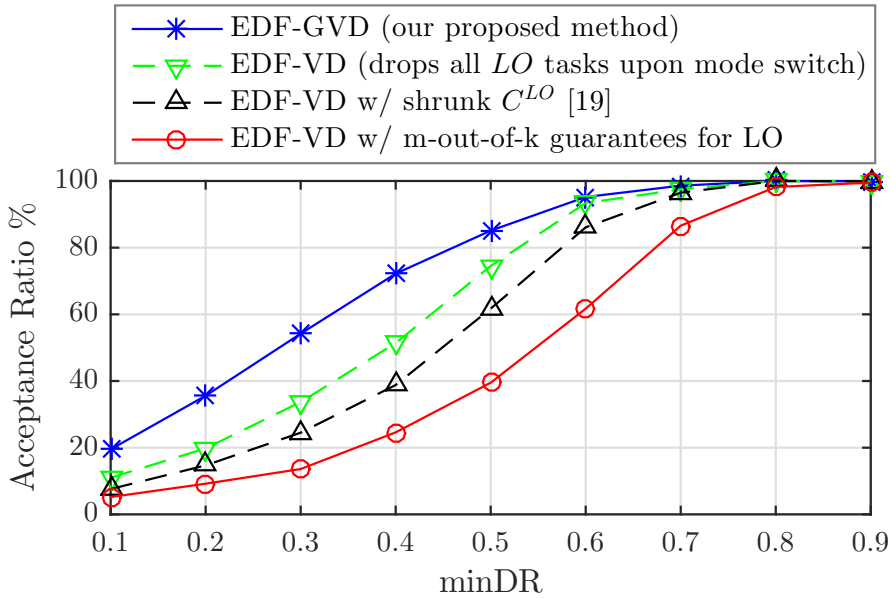
(a) Implicit deadline



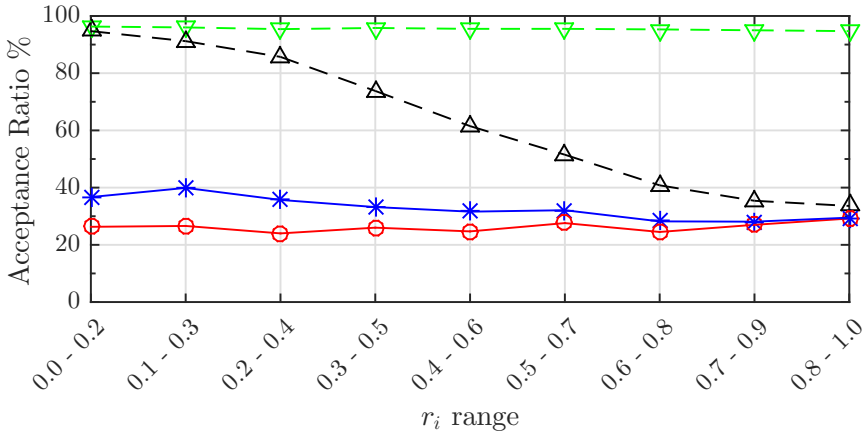
(b) Constrained deadline with $minDR = 0.5$

Figure 4.2. Variation of task set acceptance ratio for varying average utilization U

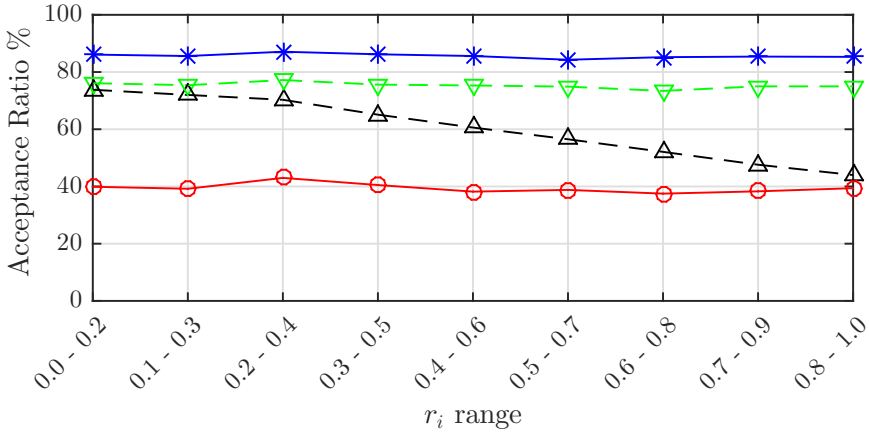
We choose the *Average Utilization* from 0.05 to 0.95 at a step size of 0.05. For every average utilization, 1000 tasks are generated. The *Acceptance Ratios* (ratio of successfully schedulable task sets and total task sets) are presented in Figure 4.2. Figure 4.2a presents the result for implicit-deadline task sets while Figure 4.2b is for constrained ones.



(a) Effect of *minDR* (U = 0.5)



(b) Effect of r_i values for implicit deadline (U = 0.7)



(c) Effect of r_i values for constrained deadline (U = 0.5)

Figure 4.3. Effect of parameters on acceptance ratio

Table 4.3. Breakdown of virtual deadline selections, mode switch upper bounds, and task set sizes

Utilization	0.4	0.5	0.6	0.7	0.8	0.9
AR (simple D) %	91.6	65.7	19.3	2.7	0.3	0.1
AR (full) %	99.3	86.2	44.9	7.1	0.7	0.3
$Avg.\ L_1/T_{max}$	0.45	0.61	0.78	0.77	0.69	0.39
Avg. size of task set	6.00	6.99	7.93	9.12	10.74	12.01

4.3.2. Observation. In Figure 4.2, we have shown the performance of EDF-GVD in comparison with other algorithms under implicit and constrained deadline settings (refer to (Guo et al., 2018)). For constrained-deadline tasks, Algorithm EDF-GVD clearly outperforms the other algorithms, as they are designed for implicit-deadline task scheduling. We perform the schedulability test for different average utilization in the range $[0.05, 0.95]$ in interval of 0.05. We observe that up to 0.5 average utilization, all the algorithms can produce the highest acceptance ratio. However, for the average utilization over 0.5, EDF-GVD delivers better acceptance ratio than all the other algorithms. Even for implicit-deadline task sets, EDF-GVD still performs better than EDF-VD with the same guarantees.

We present two virtual deadline setting mechanisms in Section 4.2.1. The first two rows of Table 4.3 report the percentage of accepted task sets under each utilization setting for the easy virtual deadline setting and the one further coped with a time-consuming binary search. It is clear that the simple heuristic can only identify a small portion of all EDF-GVD schedulable tasks, especially when utilization is high. In the table, AR refers to the acceptance ratio.

Section 4.2.3 presents a upper bound L_1 for mode switch back to LO mode (when idle occurs). The third row of Table 4.3 report the relationships between task period upper bound (T_{max}) and the bound L_1 while the last row report the average size of the task set.

In Figure 4.3a, we have shown the variation of acceptance ratio with respect to $minDR$. As $minDR$ increases towards 1.0, the constrained deadline model becomes implicit deadline model. Because it becomes almost implicit and the average utilization $U = 0.5$, the

algorithms guarantee close to 100% of the task sets. On the other hand, when $minDR$ values are less than 0.5, the constrained deadline tasks dominate the task set and our algorithm performs better than other algorithms.

In Figure 4.3b and 4.3c, the result for acceptance ratio with respect to a range of minimum degradation rate r_i for both implicit (see Figure 4.3b) and constrained-deadlines (see Figure 4.3c) is presented. The interval of the r_i range is equal for all case while the minimum bound is increased. For implicit-deadline tasks EDF-VD performs the best as it schedules only the HI-criticality tasks in HI mode. For low values of r_i, most LO-critical tasks are dropped in HI-mode and the performance of EDF-VD with shrunk C^{LO} meets EDF-VD's performance. Our algorithm provides a consistent acceptance ratio throughout the different range of r_is. For constrained deadline task sets, algorithm EDF-GVD clearly outperforms the other algorithms. It provides a consistent higher acceptance ratio throughout the different ranges of r_i.

5. PROBABILISTIC MULTIPROCESSOR SCHEDULING

In Section 4, we study an MC scheduling with graceful degradation with a specific minimum cumulative completion rate r_i for LO-criticality tasks. Under HI-criticality mode, instead of fully dropped, the LO-criticality tasks execute by following the corresponding r_i value. While this method follows a discreet scheduling procedure, in this section, we study a probabilistic MC scheduling technique. With a specific accepted failure probability F_S, our scheduler guarantee the proper scheduling of the MC task set such that the overall system failure doesn't exceed F_S. This approach provides the graceful degradation to LO-criticality tasks in a probabilistic way. While this concept was first introduced in (Guo et al., 2018), which only focus on the probabilistic schedulability on uniprocessor platforms, we further extend their work to multiprocessor platforms.

5.1. BACKGROUND AND PRELIMINARY WORK

In (Guo *et al.*, 2015), Guo et al. first proposed a novel MC system model which acknowledge the system failure probability and probabilistic worst-case execution time (pWCET) and presented an efficient MC scheduler which provides better schedulability and QoS. They considered solving the scheduling problem of independent implicit-deadline sporadic task system on uniprocessor platforms. In this thesis, we extend their work and propose an efficient multiprocessor scheduling algorithm. To the best of our knowledge, this work is the first of its kind where multiprocessor probabilistic MC scheduling is considered. Alongside the independent task set, we also provide scheduling solution for a specific type of failure dependency among the task. Before detailing our proposed method, we feel the necessity to briefly describe the work presented in (Guo *et al.*, 2015).

5.1.1. Probabilistic Schedulability on Uniprocessor Platforms. (Guo *et al.*, 2015) proposed an algorithm to schedule MC task set on uniprocessor platforms by acknowledging the failure probability of the system. In their work, they proposed a modified system model as described in Section 3.3. A failure probability parameter f_i is added to HI-criticality tasks which denote the probability that the actual execution of that task exceeding its LO-WCET. It also considers a system-wide permitted failure probability F_S. According to their work, the definition of probabilistic schedulability is described as follows:

Definition 3. (Probabilistic schedulability.) *An MC task set is strongly probabilistic schedulable by a scheduling strategy if it possesses the property that upon execution, the probability of missing any deadline is less than F_S . It is weakly probabilistic schedulable if the probability of missing any HI -criticality deadline is less than F_S. (In either case, all deadlines are met during system runs where no job exceeds its LO -WCET.)*

According to the definition of probabilistic schedulability, when the test result is strongly schedulable, then all jobs meet their deadlines with a probability of at least $1 - F_S$. And if the test result returns weakly schedulable, it guarantees that the probability of HI-criticality jobs meeting their deadlines is no less than F_S. However, if all job finishes before their LO-WCET budget, all deadlines are guaranteed to be met.

5.1.2. The LFF-Clustering Algorithm. To provide the probabilistic schedulability, (Guo *et al.*, 2015) proposed a heuristic-based clustering technique. They proposed the LFF-Clustering Algorithm which divides the HI-criticality tasks into different groups by calculating the failure probabilities in a way such that the overall system failure probability F_S doesn't exceed upon executing the clusters. We feel the necessity to briefly describe the LFF-Clustering algorithm as we use this algorithm in our work which extends the work of (Guo *et al.*, 2015) for multiprocessor platforms.

The LFF-Clustering algorithm groups all HI-criticality tasks into M number of clusters $G_1, G_2, \ldots, G_M$ based on their additional utilization cost δ_i and failure probability f_i. The additional utilization cost is the additional budget allocation of each Hi-criticality

task after a mode switch. Thus δ_i can be represented as:

$$\delta_i = (C_i^{\text{HI}} - C_i^{\text{LO}})/p_i \tag{5.1}$$

While creating the clusters, the algorithm ensures that each cluster is created in a way, such that it always maintain the following condition:

$$g_m < F_s/M \tag{5.2}$$

Here, g_m is the failure probability in cluster m. The value of g_m is calculated based on the probability of more than one single task in a cluster exceeding their LO-WCET budget within an hour. The calculation of g_m is done by following equation:

$$g_m = 1 - \prod_{i|y_i=m} (1 - f_i) - \sum_{j|y_j=m} f_j \frac{\prod_{i|y_i=m}}{1 - f_j} \tag{5.3}$$

Here, the total number of HI-criticality tasks is denoted by n_{HI} and $y_i \in \{1, 2, \ldots, M\}$ denotes to the assigned cluster number of task τ_i. In the above equation, the second term of the right-hand side is the probability of no task (in the cluster) exceeding its LO-WCET, and the last term represents the probability of exactly one of the tasks exceeding its LO-WCET in an hour. Proof has been given in (Guo *et al.*, 2015) to show that if Condition. 5.2 is maintained, then the overall system failure probability is less than F_s.

5.1.3. Runtime Strategy. For executing the LO-criticality tasks, a HI-criticality server τ_s with period 1 is allocated with utilization equal to Δ, where:

$$\Delta = \sum_{i=1}^{M} \Delta_i \tag{5.4}$$

The Δ_i is the additional utilization of each cluster and is defined by:

$$\Delta_m = \max_{i|\tau_i \in G_m} \delta_i \tag{5.5}$$

here only the maximum utilization cost in a cluster is considered because the the clusters are created in a way, such that there can be at most one task failure.

All the HI-criticality tasks along with the server τ_s are executed by following the EDF scheduling technique. At any time instant that the server is executing, if there is any active HI-criticality job (which is released but not completed yet) with the earliest deadline, those are considered for execution. Otherwise, the current job of the server is dropped. A job is dropped at its deadline if it doesn't complete within its deadline.

5.1.4. Schedulability Test. The schedulability test presented in (Guo *et al.*, 2015) provides the following schedulability conditions:

- The MC task system is strongly probabilistic schedulable if and only if the following condition holds:

$$\sum_{i=1}^{n} \frac{C_i^{\text{LO}}}{T_i} + \Delta \leq 1 \tag{5.6}$$

- If the task system is not strongly schedulable, it still can be weakly schedulable if the following conditions hold:

$$\sum_{i|\chi_i=\text{HI}} \frac{C_i^{\text{LO}}}{T_i} + \Delta \leq 1 \tag{5.7}$$

and

$$\Delta\dot{(}1 - \sum_{i|\chi_i=\text{HI}} \frac{C_i^{\text{LO}}}{T_i}) + \sum_{i=1}^{n} \frac{C_i^{\text{LO}}}{T_i} \leq 1 \tag{5.8}$$

Please refer to (Guo *et al.*, 2015) for the proofs of the above schedulability conditions.

5.2. PROBABILISTIC SCHEDULING ON MULTI-PROCESSOR PLATFORMS

With the advancements in technology, multi-processor devices are becoming more and more popular. Along with that, it is becoming more efficient to schedule real-time tasks in multi-processor platforms to get better throughput. The motivation behind the probabilistic scheduling of real-time tasks on uniprocessor platform is the same for multi-processor platforms as well. Considering the pragmatic application of applying probabilistic scheduling, by implementing a similar technique in multi-processors will dissipate the pessimistic assumption of existing scheduling mechanism and will improve the efficiency with respect to task scheduling dramatically. In this section, we propose a multi-processor scheduling technique of MC tasks considering the failure probability as proposed in Section 3.3 based on partitioned-based scheduling technique.

The partitioned-based scheduling of implicit-deadline sporadic task system can be converted into a bin-packing problem (Korte and Vygen, 2012). Hence, each processor is modeled as a bin of capacity one, and each task τ_i has the capacity of size u_i (its utilization). As bin packing problem is a combinatorial NP-Hard (Korte and Vygen, 2012), heuristics are applied to solve such problems. Similar heuristics can be applied for solving the partitioned-based scheduling problems. As our system model considers MC tasks with failure probability, we need to modify the task-sets to fit into a traditional bin packing heuristics. Here we have briefly discussed three most-common heuristics (First-Fit, Best-Fit, and Worst-Fit), then we present our algorithm to schedule our system model in multi-processor environments.

5.2.1. Different Scheduling Heuristics. For partitioned scheduling, most common heuristics are First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF). Considering the tasks are ordered by following a specific rule and every processor is indexed from 0 to upward, While scheduling with FF, the algorithm chooses the processor with the smallest index. The BF algorithm allocated the tasks to a processor with the maximal utilization. And the last

algorithm WF chooses the one with the minimal utilization. For a specific order of tasks with the utilization values 0.2, 0.6, 0.4, 0.7, 0.1, 0.3 respectively, the task allocations with different algorithms are shown in Figure 5.1.

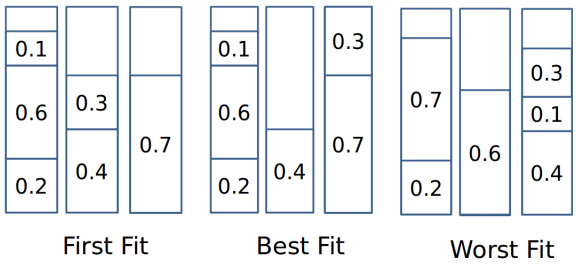


Figure 5.1. Different partitioning heuristics example

Note that, in our example if we sort the task in the decreasing order of their utilization value, we would get Reasonable Allocation Decreasing (RAD) algorithm. The three algorithms discussed above would have been called First-Fit Decreasing (FFD), Best-Fit Decreasing (BFD), and Worst-Fit Decreasing partitioned algorithm. RAD algorithms are proved to provide optimal utilization bound (López *et al.*, 2004). Hence, in our work, we consider allocating the tasks using the RAD algorithms.

5.2.2. Algorithm to Schedule MC Tasks with Failure Probability. To schedule tasks in our proposed system model in multiprocessor platforms, we present a slightly different scheduling approach than the uniprocessor scheduling prested in (Guo *et al.*, 2015). Initially, all the tasks are grouped into different clusters using the same LFF-Clustering algorithm presented before. Then we schedule the clusters in different processors by using different RAD partitioned heuristics. Note that, for every cluster, there is an additional utilization cost Δ_m which is also needed to be allocated in case any task of the cluster exceeds is LO-criticality WCET. For every processor, we need to allocate a server which has a utilization equal to the sum of additional utilization cost Δ_m of all the clusters allocated in that processor. For example, While considering of scheduling three clusters on two processors and if the Clusters 1 and 2 are allocated on Processor 1 and the Cluster 3 is

allocated on Processor 2, then we also need to allocated a server with utilization $(\Delta_1 + \Delta_2)$ in Processor 1, and another server with utilization Δ_3 in the Processor 2. So while applying the partitioned heuristic, we need to accommodate the demand of server utilization as well.

5.2.3. Scheduling LO-Criticality Tasks. The LO-criticality tasks are considered for allocation once all the HI-criticality clusters and $\Delta's$ are allocated. The LO tasks are allocated using the same partitioning techniques used for the HI tasks.

5.3. SCHEDULABILITY

The schedulability of the task set is determined in two different steps. First, we have to check whether the tasks can be properly allocated to the available processors. Upon successful allocation, we need to check whether the MC task systems can be scheduled runtime even under worse conditions. Also in each step, there is a possibility that either only HI tasks or all the tasks are allocated/schedulable. Based on the allocation the tasks set can be strongly allocated or weekly allocated. If all the clusters, $\Delta's$, and LO-tasks are allocated then we call it strongly allocated task set. If only the clusters and $\Delta's$ are allocated properly but not the LO-tasks then we call it weakly allocated task set. Otherwise, the task sets cannot be considered for further schedulability test. Upon successful allocation, we need to check the schedulability of the allocated task set on all the processors. The schedulability test is done by following the similar technique used for the uniprocessor scheduling.

The following theorems are used to check the successful allocation and the schedulability of the task sets. Before going into the schedulability conditions, it is necessary to introduce the parameter α and β. α is the utilization factor of a task set, i.e., the maximum utilization among all tasks. β is the maximum number of tasks of α which fit into one processor under EDF scheduling. β can be expressed as a function of α

$$\beta = \lfloor \frac{1}{\alpha} \rfloor \tag{5.9}$$

Lopez et al. (López *et al.*, 2004) proved that for multiprocessor partitioned scheduling using EDF, FFD, BFD, and WFD algorithms provide the optimal upper bound, which is the following:

Lemma 6. *If $m > \beta n$ then $U(n, \alpha) = \frac{\beta n+1}{\beta+1}$*

Here, $U(n, \alpha)$ denotes the utilization bound to schedule n number of tasks using RAD algorithms in m processors, where the utilization factor for the task set is α.

5.3.1. Task Allocation Conditions. For the HI-tasks we need to allocate each cluster with its Δ_m in the same processor as in the partitioned scheduling, a task is always needed to be executed on the same processor which it was initially allocated. Lets assume there are m clusters with utilization $UC_1, UC_2, \ldots, UC_m$ with the corresponding delta values as $\Delta_1, \Delta_2, \ldots, \Delta_m$. As each cluster is needed to be allocated to a processor with its corresponding Δ, let assume α_h is the utilization factor of the sum of the cluster's utilization and it's Δ value, and β_h is the maximum number of tasks of α_h which fit into one processor under EDF scheduling.

$$\alpha_h = \max_{i \in 1,2,\ldots,m} (UC_i + \Delta_i) \tag{5.10}$$

$$\beta_h = \lfloor \frac{1}{\alpha_h} \rfloor \tag{5.11}$$

Similarly, lets n defines the number of LO-tasks in the system, and let α_l and β_l define the utilization factor and the β value for the LO-tasks respectively. Let α_s be the utilization factor of the total system, which is the maximum of α_h and α_l. Furthermore, β_s represents the β value of the system (i.e., the HI-criticality clusters and the LO-criticality tasks).

$$\alpha_s = \max(\alpha_h, \alpha_l) \tag{5.12}$$

$$\beta_s = \lfloor \frac{1}{\alpha_s} \rfloor \tag{5.13}$$

Theorem 1. *All the* HI*-criticality clusters along with their server allocation (Δ) and all the* LO*-criticality tasks can be allocated (i.e., strongly allocated task-set) to p processor if the following two condition holds,*

$$p > \beta_s(m + n) \tag{5.14}$$

$$U_s \leq \frac{\beta_s(m + n) + 1}{\beta_s + 1} \tag{5.15}$$

here U_s is the sum of the utilization of all cluster, all LO*-criticality tasks, and the Δs.*

Proof. Here, each cluster and LO-criticality tasks can be seen as a single entity with specific utilization demand. The total number of entity here is $(m + n)$. Thus according to Theorem 4, the maximum utilization bound can be $\frac{\beta_s(m+n)+1}{\beta_s+1}$. So for a successful allocation, the system utilization U_s must be no greater than the utilization bound. □

Theorem 2. *All the* HI*-criticality clusters along with their server allocation (Δ) can be allocated (i.e., weakly allocated task-set) to p processor if the following two condition holds,*

$$p > \beta_h m \tag{5.16}$$

$$U = \frac{\beta_h m + 1}{\beta_h + 1} \tag{5.17}$$

Proof. Theorem 6 can be proved similar to Theorem 5 by using the utilization bound presented in Theorem 4. □

5.3.2. Task Schedulability Condition. Upon successful allocation (either strongly or weakly allocated), the schedulability of the task set can be determined by running the pMCMP (probabilistic Mixed-Criticality on MultiProcessor) algorithm (presented in 4). pMCMP algorithm basically use pMC algorithm presented in (Guo *et al.*, 2015) on all the processors to check the schedulability. The result of pMCMP can be strongly schedulable, weakly schedulable, or non-determined. If a task set is only weakly allocated on the available processors, the task set can never be considered as strongly schedulable.

Algorithm 4: pMCMP algorithm

```
Data: Allocation of HI-criticality Δ_i s and LO-criticality task-set τ^i on each processor p_i
Result: The schedulability of the task set
if ∀p_i, pMC returns strongly-schedulable then
  | return strongly-schedulable;
end
else if ∀p_i, pMC returns weakly-schedulable then
  | return weakly-schedulable;
end
else
  | return non-determined;
end
```

Theorem 3. *The scheduled task set satisfies the correctness criteria of probabilistic schedulability presented in (Guo* et al.*, 2015) for multiprocessor platforms. i.e., the schedulability test pMCMP for multiprocessor scheduling is sufficient in the following sense:*

- *If it returns strongly probabilistic schedulable, the probability of any task missing its deadline is no greater than F_S*
- *If it returns weakly probabilistic schedulable, the probability of any* HI*-criticality task missing its deadline is no greater than F_S, and no deadline is missed when all jobs finish upon execution of their* LO*-WCETs.*

Proof. After a successful allocation, each processor has a specific set of Δs and LO-criticality tasks assigned for scheduling. Let $n_1, n_2, \ldots, n_M$ denote the number of clusters assigned to p processors (note that there are total M number of clusters). Hence, each processor assignment can be seen as a subset problem of uniprocessor scheduling presented in (Guo *et al.*, 2015).

For processor 1, similar to the proof of Eqn. 5.2 (Guo *et al.*, 2015), we can show that the failure probability of processor 1 is no greater that $(n_1 \times F_S)/M$. Similarly, for other processors the failure probability lower bounds are $(n_1 \times F_S)/M, (n_2 \times F_S)/M, \ldots, (n_M \times F_S)/M$ respectively. As the tasks are independent. The total failure probablity of the system is no greater than $\sum_{i=1}^{M}(n_i \times F_S)/M = F_S$. □

5.4. CONSIDERING COVARIANCE/FAILURE DEPENDENCY

In our previous model, we have considered independent tasks only, e.g., the failure probability f_i of each task τ_i is independent of each other. As a result, whether a HI task fails to complete within its LO-WCET budget does not affect the completion of other HI tasks. In traditional models, it is assumed that all the HI tasks exceed their LO-WCET budget at the same time. That means failure probabilities of all HI tasks are dependent on each other, which is rather a pessimistic assumption. However, in practical systems, not every failure probability is independent. It is possible that while most of the HI tasks are independent, a certain amount of task is directly dependent to each other with respect to their probability of exceeding LO-WCET, i.e., once a HI task exceeds the LO-WCET, other dependent tasks also exceed their LO-WCET budget, regardless of their own failure probability. By considering this scenario, We propose an extended model in this section by introducing the covariance/failure-dependency in our task model.

Table 5.1. Sample covariance matrix of an MC task set with 8 tasks

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
τ_1	-	0	0	0	1	0	0	0
τ_2	-	-	1	0	0	1	0	0
τ_3	-	-	-	0	0	1	1	0
τ_4	-	-	-	-	0	0	0	0
τ_5	-	-	-	-	-	0	0	1
τ_6	-	-	-	-	-	-	0	0
τ_7	-	-	-	-	-	-	-	0

5.4.1. Covariance Matrix. We introduce a covariance matrix where the dependency between each tasks failure to complete before LO-WCET. In Table. 5.1, a sample covariance matrix is shown for eight HI-criticality tasks. Here the covariance is represented by either 1 or 0. 0 represents the independence between two tasks while the 1 represents that there is a dependency between the failure probability of two tasks. Note that, for sim-

plicity, we only consider 0 and 1 values for covariance. As a result, if there is a dependency between two tasks, we assume that upon exceeding the LO-WCET budget of one task, the other dependent task/s will also exceed their LO-WCET budget simultaneously.

Definition 4. (Failure-dependent tasks.) *Two tasks are considered failure-dependent when upon scheduling those tasks on the same processor, if one of those tasks exceeds its* LO-*WCET, the other one also exceeds. So, both of the tasks exceeds their* LO-*WCET simultaneously. In Table 5.1, if there is a 1 between two tasks, those tasks are failure-dependent. For example, τ_2 and τ_3 are failure dependent but τ_1 and τ_2 are not. Note that, two tasks are failure-dependent only if those are scheduled on the same processor. Upon scheduling on different processors, those tasks can be executed independently.*

5.4.2. Task Isolation using Graph Model. Similar to the clustering problem without covariance, finding the optimal clustering for the new problem is also NP-Hard. Thus, we propose a heuristic to find an efficient solution to the problem. From the covariance matrix, we can visualize a task set as a graph problem. We can assume the tasks as a node of the graph and the failure dependencies as edges between the nodes, i.e., if there is a 1 between two tasks, we can consider an edge between those two nodes (tasks). Note that, if the graph is fully connected, the problem will become a traditional MC scheduling problem as in our strategy, we will need to create a cluster for each task. In practical scenarios, the graph is assumed to be a disconnected graph as there can be both independent and failure dependent tasks in a system. Hence there will be multiple islands in the graph which consist of interdependent tasks. Example 5.4.1 demonstrate the graph transformation of the covariance matrix.

Example 5.4.1. *Consider the covariance matrix for all* HI-*criticality tasks of a task set is shown in Table 5.1. Here from the definition of covariance matrix, we can see that the failure of τ_1 is dependent on τ_5, and the failure of τ_5 is dependent on τ_8. Hence to transform the covariance matrix into a graph, we can consider τ_1, τ_5, and τ_8 as the node of the graph*

and put an edge between τ_1 and τ_5. Similarly, we put another edge between τ_5 and τ_8. By following this transformation rule, we can get a disconnected graph with three islands as shown in Figure 5.2. Here each island represents the inter-dependent tasks. Note that, an island can consist of only a single task (τ_4 in this case) if it is fully independent.

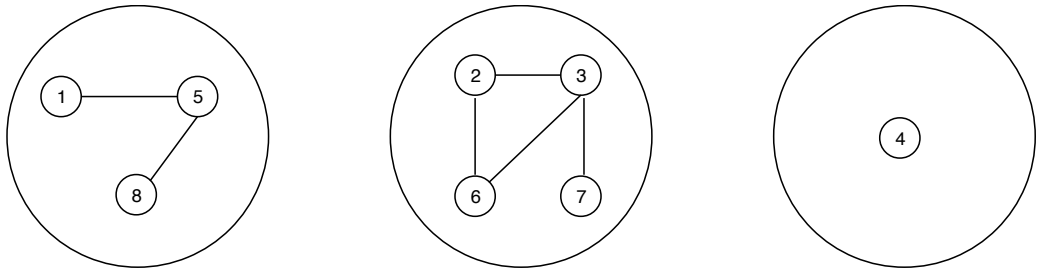


Figure 5.2. Graph transformation of the covariance matrix

Definition 5. (Transitive failure-dependency.) *If two tasks are not directly failure-dependent to each other but they have a common failure-dependent task, then we call the failure-dependency between the first two tasks transitive failure-dependency. In Figure 5.2, τ_1 and τ_8 are not directly failure-dependent (i.e., independent tasks), but they both have failure dependency with τ_5. Hence, if we schedule these three tasks together, they will exceed their* LO-*WCET budget simultaneously. However, if we schedule τ_5 in a separate processor, τ_1 and τ_8 will have no failure-dependency and can act as independent tasks.*

Upon transforming the covariance matrix into a graph, we apply our clustering heuristics. In the previous LFF-Clustering algorithm we sorted all the tasks in descending order based on their additional utilization δ_i value and we keep adding the tasks one by one to clusters until the Equation 5.2 is not violated. While scheduling the task set including failure-dependent tasks, we isolate the tasks of each island into m (number of processors) number of groups in a way such that there are no failure-dependent tasks in any group. By doing that, we ensure that no two inter-dependent tasks are grouped into a single processor.

ISOLATING TASKS OF EACH ISLAND: Once we convert the task set as a graph with different islands, we create m number of groups of independent tasks. We can visualize this problem as a $m - coloring$ graph problem. In a $m - coloring$ problem, the nodes of a graph are colored with at most m number of colors where no two nodes are adjacent to each other (i.e., there is no edge between them). If we can properly color the graph with m or less number of colors then we can easily allocate each colored nodes to distinct processors. However, the $m - coloring$ problem is an NP-Complete problem (Irving, 1983). As a result, we will need an approximate algorithm to solve such problems. Furthermore, it may not always be possible to color the subgraph in each island with $m - colors$ by using the approximate algorithm (even by using an optimal algorithm). Hence, we propose a modified approximate algorithm to color each subgraph with m colors.

To accomplish this goal, we use a modified greedy coloring algorithm. As our base greedy coloring algorithm, we use The Welsh Powell algorithm (Welsh and Powell, 1967) also known as Largest-First (LF) coloring algorithm. In LF algorithm, the vertices are sorted in non-increasing order of their degrees (number of edges). Then at each step, the nodes with largest degrees and its non-adjacent nodes are colored with the same color. This procedure is done repeatedly until all the nodes are colored. We choose this heuristic because, if we isolate the nodes with the highest degrees first, we can eliminate the largest number of transitive failure-dependencies. Then we gradually reduce the number of transitive failure-dependencies and increase the number of independent tasks.

So we start coloring from the node with the highest degree and its adjacent nodes. Then we delete the edge of all the colored nodes. By doing this, we remove the dependencies of other nodes with the colored nodes. However, when we already use $m - 1$ colors and only one color is left to use, we need to contract (merge) the remaining failure-dependent tasks. When at the last processor, there are still some nodes which are connected (failure-

dependent), we contract those nodes in a single node and use the u_i^{LO} and δ_i of the node as the sum of the corresponding values of all the connected nodes. The steps of the coloring technique are shown in Algorithm 5 and further demonstrated in Example 5.4.2.

Algorithm 5: $m - coloring$ algorithm

```
Data: G = {V, E}, m
Result: m − colored graph
Sort all nodes V_i s in non-increasing order or their degrees;
for i ← 1 to m do
    /* All nodes are colored                                    */
    if ∀V_i is colored then
        return;
    end
    /* If at last processor, there exists some connected nodes  */
    if (i == m) and (∃E_i) then
        forall Connected subgraph do
            V_new ← {connected_nodes};
            V_new.u_i^LO ← {connected_nodes}.u_i^LO;
            V_new.δ_i ← {connected_nodes}.δ_i;
        end
    end
    else
        Color the nodes following LF (Welsh and Powell, 1967) Algorithm;
    end
end
```

Example 5.4.2. *In Figure 5.3, the tasks of one island is shown. Here, we need to allocate the tasks on two processors, i.e., we have to color the graph with 2 colors. To do this, we first take the node with the highest degree (τ_4 with degree 4) and color it with green. Then we color the non-adjacent nodes of τ_4 (i.e., τ_1, τ_2, and τ_8 with green and remove the edges of those nodes. We can no longer use green in this graph. Now we have one processor (color) left but there is still two tasks τ_6 and τ_7, which are failure-dependent to each other. So, we merge these two tasks and all the nodes become independent. Finally, we color all the nodes with blue (τ_3, τ_5, and τ_{6+7}).*

5.4.3. Task Allocation and Scheduling. To schedule the task set, we first allocate the tasks by leveraging the above-mentioned techniques. We first run DFS over the covariance matrix and convert them to different number of islands. Then we use the $m - coloring$

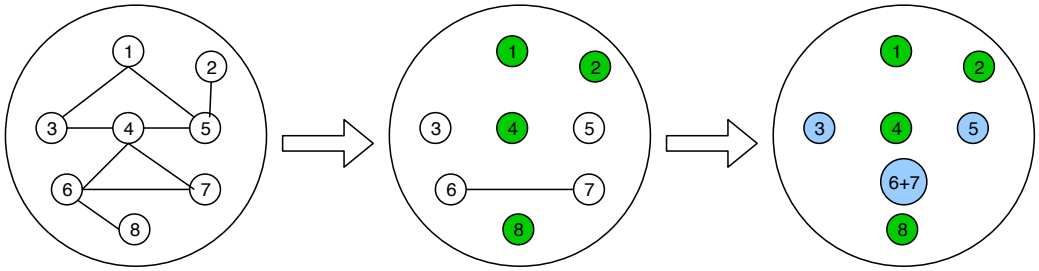


Figure 5.3. Graph coloring with $m = 2$

algorithm presented in Algorithm 5 to color the nodes of each island with at most m number of distinct colors. Then the nodes in each island are sorted in descending order with respect to their δ_i values and the islands themselves are then sorted in descending order based on $max(\delta_i)$ values of the member nodes. HI-criticality tasks are allocated by following rules:

- First we assign the tasks of each color of each island by to a distinct processor by following the WF heuristics on processor capacity.
- While assigning a task to a processor, we create a cluster following the LFF-Clustering algorithm. First, we keep adding the same colored tasks in an island to the existing task/s assigned to that processor. If we cannot assign the new task to an existing cluster, we create a new cluster. Once all nodes of the same color are allocated, we allocate the next color tasks to a different processor.
- Every time we add a new task to a processor, we update the Δ value of the processor.
- Once all islands with multiple tasks are assigned, we assign the remaining single-task islands by following the WF partitioning heuristic.

If all the tasks are allocated, the task set becomes strongly allocated, while only the allocation of HI-criticality tasks results in a weakly allocation. Upon successful allocation, we run the pMCMP algorithm to check the schedulability of the task set. The task allocation procedure is presented in Algorithm 6.

Algorithm 6: Task Allocation Algorithm with Covariance

```
Data: F_S, {f_i}_{i=1}^{nHI}, {u_i^LO}_{i=1}^{n} {δ_i}_{i=1}^{n}, covariance matrix
Result: Task allocation result
Run DFS on covariance matrix and get islands {I_j}_{j=1}^{k};
Color the nodes of each island using Algorithm 5;
Sort ∀τ_i ∈ ∀I_j in descending order w.r.to δ_i;
Sort ∀I_i based on max(δ_i) ∈ I_i;
P = {p_1, p_2, ..., p_m};
forall multi-task islands I_i do
    allocated = φ;
    forall ∀τ_j ∈ I_i do
        allocate τ_j into p_k ∈ {P − allocated} following WF and update Δ_k;
        allocated = allocated ∪ p_k
    end
end
if all LO tasks are allocated using WF then
    return strongly − allocated;
end
return weakly − allocated;
```

5.5. EXPERIMENTAL EVALUATION

In this section, we present extensive experimental evaluations to show the performance of algorithm pMCMP. To the best of our knowledge, our work is the first to propose probabilistic MC scheduling on multiprocessor platforms. As a result, there is no baseline to be compared with. We have performed a number of experiments by varying different important factors to observe the efficiency of our algorithm.

5.5.1. Workload Generation. To conduct the experiments, we have generated MC tasks based on the following parameters.

- M : The number of processor cores.
- U_a : The average utilization for the task set. The average is calculated by averaging the LO and HI-criticality utilization of the task set.
- $P_{\text{HI}} = 0.5$: The probability of a task to be a HI-criticality one.

- $R = 4$: Denotes the maximum ratio of u_i^{HI} to u_i^{LO}. u_i^{HI} is generated uniformly from $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

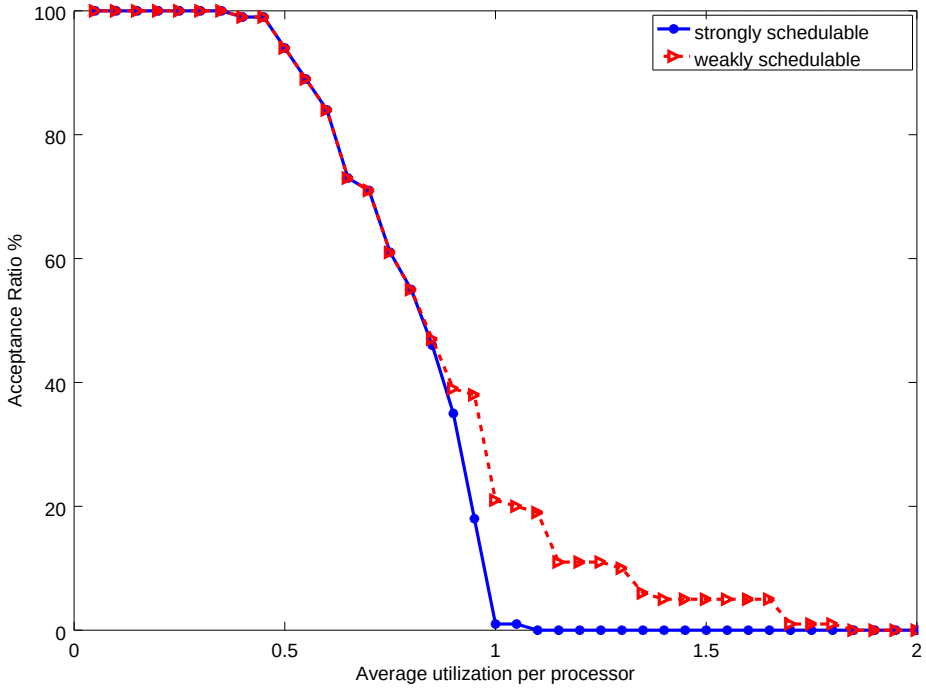
- F_S: The system-wide permitted failure probability. We use 10^{-6} as the value of F_S for all of our experiments.

We performed the simulation for average utilization ranging from $0.05M$ to $2M$ with increasing at step size $0.05M$. For every average utilization, we generate 100 task sets which consist of 20 tasks each. Note that, for most of the experiments, we have measured the performance with respect to average utilization as we wanted to show the improved quality of service for normally generated MC task sets.

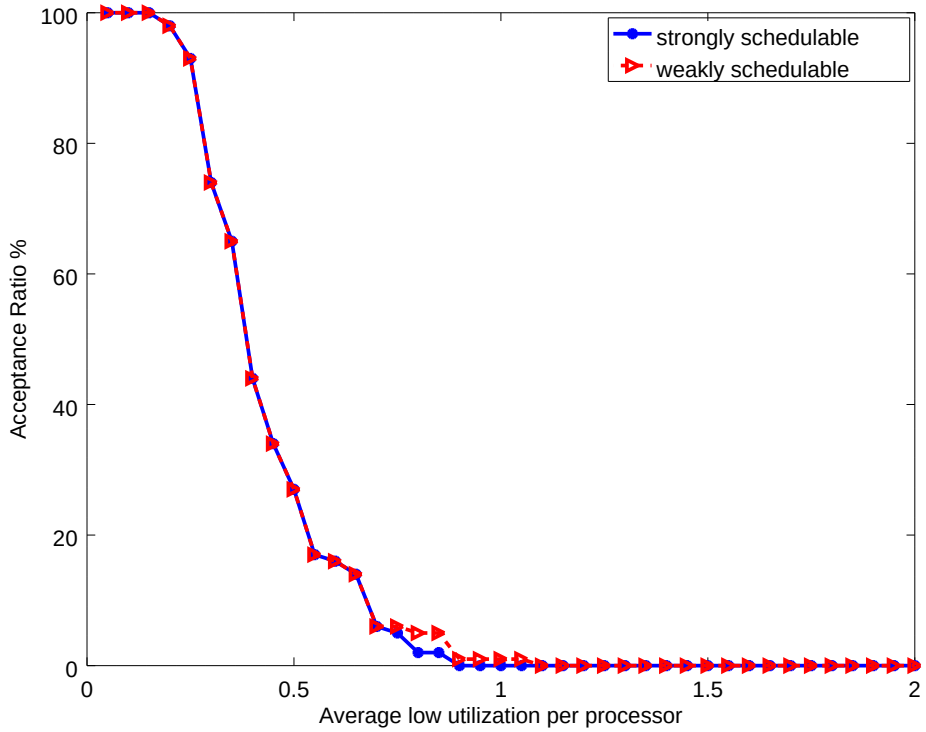
At first, for a specific average utilization, we use *UUniFast algorithm* Bolado *et al.* (2004) to generate a lognormal distribution of U_a for all the tasks in a task set. The values of u_i^{LO} is uniformly generated from $[\frac{2 \times u_i^a}{R+1}, u_i^a]$ so that the value of u_i^{HI} is always in the range $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

5.5.2. Evaluation Results. We execute a set of MC tasks under our proposed algorithm by varying different parameters. Simulation results for various scenarios are presented in Figure 5.2 to Figure 5.7. We perform the following simulations:

The schedulability performance of pMCMP is shown in Figure 5.6. While the Figure 5.4a shows the acceptance ratio (ratio of successfully scheduled task sets over total number of task sets) with respect to average utilization U_a, the results in Figure 5.4a shows the acceptance ratio with respect to u_i^{LO}. In both figures, we show the acceptance ratio for both strongly schedulable and weakly schedulable task sets. In Figure 5.4a, we can see that a good number of task sets is schedulable when the average utilization of the task set is one or even higher as the average utilization is calculated based on both u_i^{LO} and u_i^{HI} but pMCMP doesn't need to allocate full HI-WCET budget. To understand the schedulability with respect to u_i^{LO}, we further performed the experiment presented in Figure 5.4b where the u_i^{LO} is generated following the lognormal distribution using the *UUniFast algorithm* Bolado *et al.* (2004) algorithm.



(a) for varying average utilization U_a



(b) for varying average low utilization U_{LO}

Figure 5.4. Acceptance ratio for pMCMP in an 4-core platform under different utilizations

Next, in Figure 5.5, we present the acceptance ratio for all three heuristics (FF,BF, and WF) discussed in Section 5.2.1. Previously we discussed that all RAD algorithms share the same utilization bound while task partitioning, and the result shows the same. Note that, we use only strongly schedulable task set to calculate the acceptance ratio from this experiment as only the strongly schedulable task sets provide the graceful degradation to LO-criticality tasks.

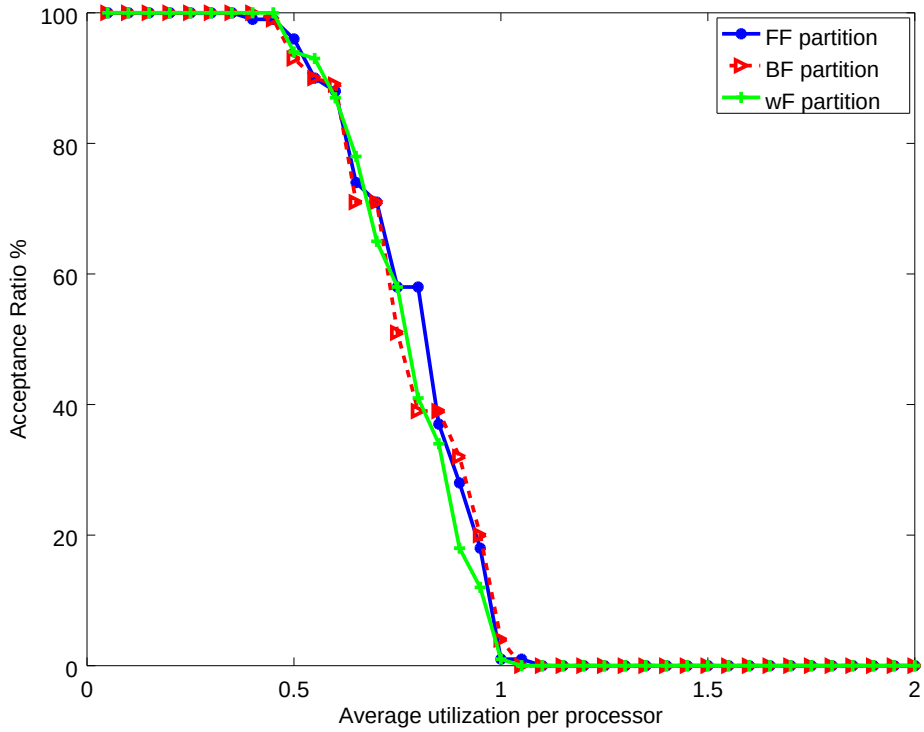
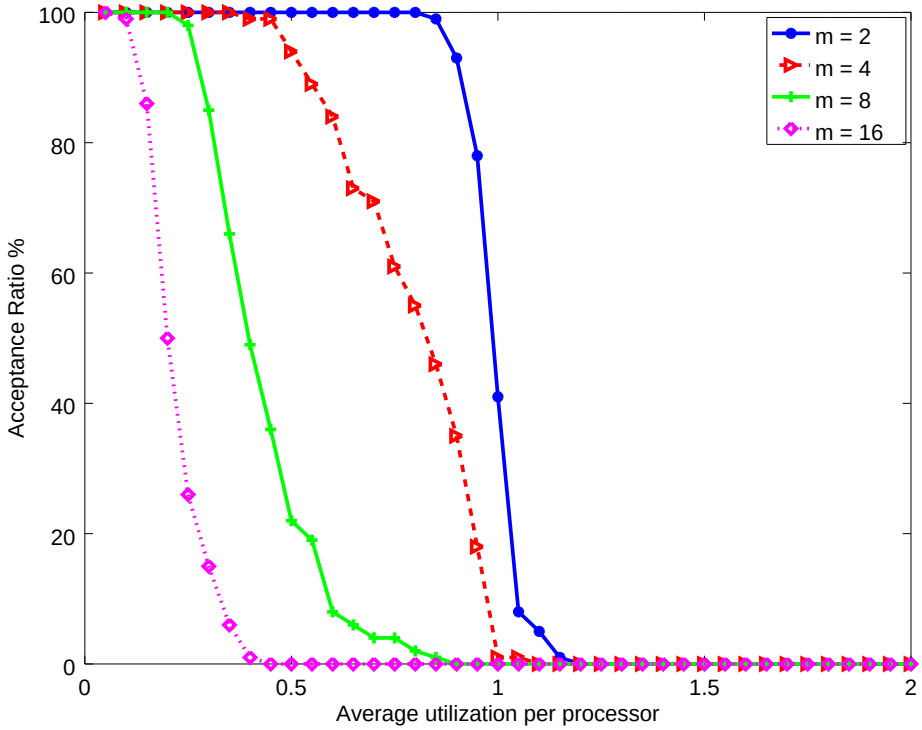
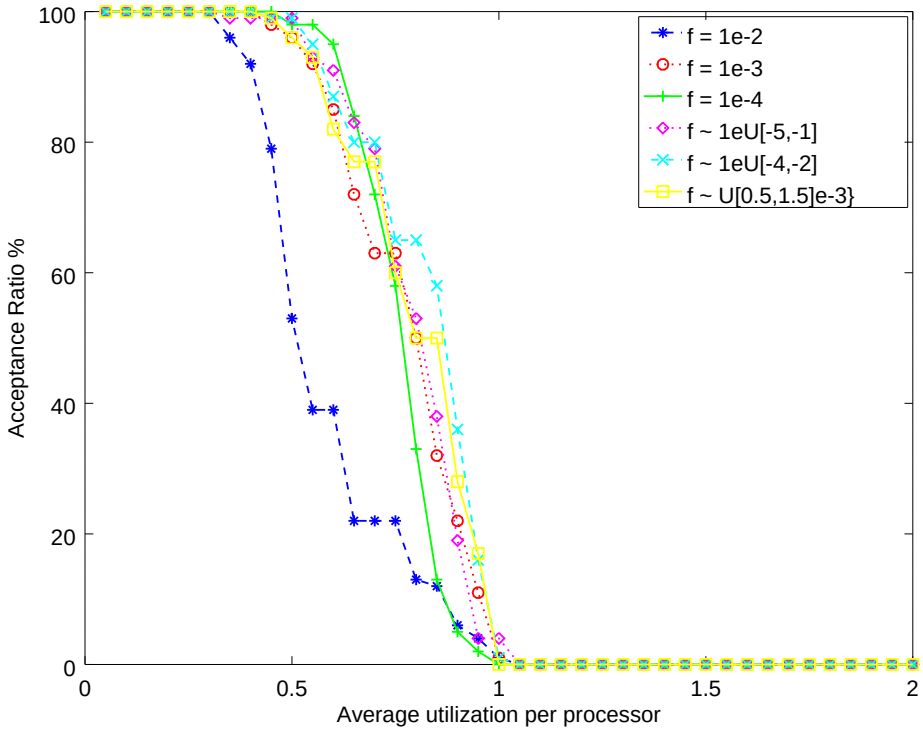


Figure 5.5. Performance of pMCMP in an 4-core platform under different partition heuristics

We also present the percentage of successful strongly schedulable task set under different number of processors (Figure 5.6a) and different number of f_i values (Figure 5.6b). As expected, the performance of schedulability decreases with the increase of the number of the processors by following the performance of the partition heuristics. On the other hand, with the lower f_i values, we get better acceptance ration as the algorithm can create more clusters and thus needs to allocate a smaller Δ.



(a) for various number of processors (m)



(b) for various distribution assumptions made to f_is

Figure 5.6. Acceptance ratio for pMCMP under various parameters

We also performed simulation on different density of covariance. The number of edges of the covariance graph (1 in covariance matrix) are randomly generated based on the density of edge. The simulation result is presented in Figure 5.7. Under lower densities, our algorithms perofmred surprisingly well. With the increase of density, the possibility of merging also increases and the acceptance ratio decreases.

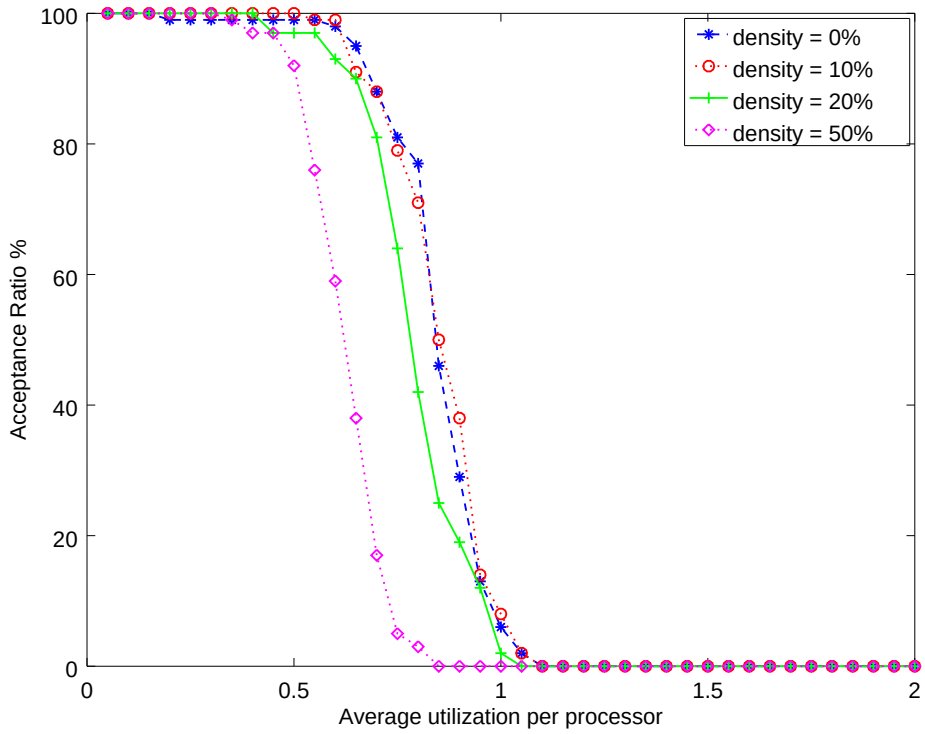


Figure 5.7. Performance of pMCMP in an 4-core platform under different density of covariance

6. CONCLUSION

The main objective of this thesis is to enable graceful degradation to mixed-criticality task systems. In this thesis, we consider two different system model to provide enhanced service to LO-criticality tasks which improve the overall quality of service of the system.

The first contribution of this thesis is the study of mixed-criticality scheduling where graceful degradation is provided in the form of a minimum cumulative completion rate. The key point of this technique is that we provide a guaranteed service to LO-criticality tasks under HI-criticality mode. Two admission-control algorithms are presented to decide the add/drops of LO-criticality jobs in HImode. We study the pseudo-polynomial time schedulability test and present a mechanism for the backward mode switch. The performance efficiency of our approach is presented in several experimental results.

Our second condition is to provide probabilistic schedulability to mixed-criticality task systems on multiprocessor platforms. It is assumed that a system-wide permitted failure probability along with the probability of each HI-criticality tasks exceeding their LO-WCET is known. Based on the assumption, we provide the necessary algorithm and schedulability analysis. This approach enhances the schedulability of LO-criticality tasks as it doesn't drop them until the permitted failure probability is violated. We further consider the failure dependencies between tasks while scheduling on multiprocessor platforms and present the detailed simulation results.

REFERENCES

Abeni, L. and Buttazzo, G., 'Qos guarantee using probabilistic deadlines,' in 'Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS'99),' 1999 .

ARINC, 'Aeronautical Radio, INCorporated,' http://www.arinc.com/, ????, accessed: July 21, 2017.

AUTOSAR, 'AUTOSAR (AUTomotive Open system ARchitecture),' https://www.autosar.org/, ????, accessed: July 21, 2017.

Awan, M., Bletsas, K., Souto, P., and Tovar, E., 'Semi-partitioned mixed-criticality scheduling,' in 'Proceedings of the International Conference on Architecture of Computing Systems (ARCS),' Springer, 2017 pp. 205–218.

Barhorst, J., Belote, T., Binns, P., Hoffman, J., Paunicka, J., Sarathy, P., Stanfill, J., Stuart, D., and Urzi, R., 'White paper: A research agenda for mixed criticality systems,' Available: https://www.cse.wustl.edu/ cdgill/CPSWEEK09$_MCAR/RBO - 09 - 130????, accessed : July 12, 2017$.

Baruah, S., 'Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors,' IEEE Transactions on Computers, 2004, **53**(6), pp. 781–784.

Baruah, S., 'A scheduling model inspired by control theory,' in 'Proceedings of the 6th International Real-Time Scheduling Open Problems Seminar,' 2015 .

Baruah, S., 'Mixed-criticality scheduling theory: Scope, promise, and limitations,' IEEE Design and Test, 2018, **35**(2), pp. 31–37.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L., 'Scheduling real-time mixed-criticality jobs,' in 'Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS),' 2010 pp. 90–101.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L., 'Scheduling real-time mixed-criticality jobs,' IEEE Transactions on Computers, 2012a, **61**(8), pp. 1140–1152.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L., 'The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems,' in 'Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS),' 2012b .

Baruah, S., Bonifaci, V., D'angelo, G., Li, H., Marchetti-Spaccamela, A., Van Der Ster, S., and Stougie, L., 'Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems,' Journal of the ACM (JACM), 2015, **62**(2), p. 14.

Baruah, S., Burns, A., and Davis, R., 'Response-time analysis for mixed criticality systems,' in 'Proceedings of the 32nd International Real-Time Systems Symposium (RTSS),' IEEE, 2011a pp. 34–43.

Baruah, S., Burns, A., and Davis, R., 'Response-time analysis for mixed criticality systems,' in 'Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS),' 2011b .

Baruah, S., Burns, A., and Guo, Z., 'Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors,' in 'Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS'16),' 2016 .

Baruah, S. K., Bonifaci, V., DâĂŹAngelo, G., Marchetti-Spaccamela, A., Van Der Ster, S., and Stougie, L., 'Mixed-criticality scheduling of sporadic task systems,' in 'European Symposium on Algorithms,' Springer, 2011c pp. 555–566.

Baruah, S. K., Mok, A. K., and Rosier, L. E., 'Preemptively scheduling hard-real-time sporadic tasks on one processor,' in 'Proceedings of the 11th IEEE Real-Time Systems Symposium,' ISBN 0818621125, 1990 doi:10.1109/REAL.1990.128746.

Bernat, G., Burns, A., and Liamosi, A., 'Weakly Hard Real-Time Systems,' IEEE Transactions on Computers, 2001, **50**(4), pp. 308–321, doi:10.1109/12.919277.

Bletsas, K. and Petters, S., 'Using NPS-F for mixed-criticality multicore systems,' in 'Proceedings of the 33rd International Real-Time Systems Symposium (RTSS),' IEEE, 2012 pp. 36–36.

Bolado, M., Posadas, H., Castillo, J., Huerta, P., Sanchez, P., Sánchez, C., Fouren, H., and Blasco, F., 'Platform based on open-source cores for industrial applications,' in 'Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings,' volume 2, IEEE, 2004 pp. 1014–1019.

Branicky, M., Phillips, S., and Wei Zhang, 'Scheduling and feedback co-design for networked control systems,' in 'Proceedings of the 41st IEEE Conference on Decision and Control,' ISBN 0-7803-7516-5, 2002 doi:10.1109/CDC.2002.1184679.

Burns, A., 'How to gracefully degrade,' 2017, keynote given at Dagstuhl Seminar 17131.

Burns, A. and Davis, R., 'Mixed-Criticality systems: A review,' Available: https://www-users.cs.york.ac.uk/burns/review.pdf, 2017a.

Burns, A. and Davis, R., 'Mixed-criticality systems: A review,' 2017b.

Burns, A. and Davis, R. I., 'A survey of research into mixed criticality systems,' ACM Computing Surveys (CSUR), 2017c, **50**(6), p. 82.

Chen, Y., Li, Q., Li, Z., and Xiong, H., 'Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling,' Chinese Journal of Aeronautics, 2014, **27**(4), pp. 856–866.

Chwa, H. S., Shin, K. G., and Lee, J., 'Closing the gap between stability and schedulability: a new task model for Cyber-Physical Systems,' in 'Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'18),' 2018 .

Cucu-Grosjean, L., Santinelli, L., Houston, M., Lo, C., Vardanega, T., Kosmidis, L., Abella, J., Mezzetti, E., Quinones, E., and Cazorla, F., 'Measurement-based probabilistic timing analysis for multi-path programs,' in 'Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS'12),' 2012 .

David, L. and Puaut, I., 'Static determination of probabilistic execution times,' in 'Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04),' 2004 .

Davis, R. I., Santinelli, L., Altmeyer, S., Maiza, C., and Cucu-Grosjean, L., 'Analysis of probabilistic cache related pre-emption delays,' Proceedings of the 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS'13), 2013.

Díaz, J., Garcia, D., Lee, C., Bello, L., López, J., and Mirabella, O., 'Stochastic analysis of periodic real-time systems,' in 'Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02),' 2002 .

Easwaran, A., 'Demand-based scheduling of mixed-criticality sporadic tasks on one processor,' in 'Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS),' IEEE, 2013 pp. 78–87.

Edgar, S. and Burns, A., 'Statistical analysis of wcet for scheduling,' in 'Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01),' 2001 .

Ekberg, P. and Yi, W., 'Bounding and shaping the demand of generalized mixed-criticality sporadic task systems,' Real-Time Systems, 2014, **50**, pp. 48–86, doi: 10.1007/s11241-013-9187-z.

Fleming, T. and Burns, A., 'Incorporating The Notion of Importance into Mixed Criticality Systems,' in 'Proceedings of the 2nd Workshop on Mixed Criticality Systems,' 2014 .

Gardner, M. and Liu, J., 'Analyzing stochastic fixed-priority real-time systems,' in 'Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99),' 1999 .

Gettings, O., Quinton, S., and Davis, R. I., 'Mixed criticality systems with weakly-hard constraints,' in 'Proceedings of the 23rd International Conference on Real Time and Networks Systems,' ISBN 9781450335911, 2015 doi:10.1145/2834848.2834850.

Gratia, R., Robert, T., and Pautet, L., 'Generalized mixed-criticality scheduling based on RUN,' in 'Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS),' ACM, 2015a pp. 267–276.

Gratia, R., Robert, T., and Pautet, L., 'Scheduling of mixed-criticality systems with RUN,' in 'Proceedings of the 20th International Conference on Emerging Technologies & Factory Automation (ETFA),' IEEE, 2015b pp. 1–8.

Griffin, D. and Burns, A., 'Realism in statistical analysis of worst case execution times,' in 'Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET'10),' 2010 .

Gu, X. and Easwaran, A., 'Dynamic Budget Management with Service Guarantees for Mixed-Criticality Systems,' in 'Proceedings of the 37th IEEE Real-Time Systems Symposium,' ISBN 978-1-5090-5303-2, 2016 doi:10.1109/RTSS.2016.014.

Guan, N., Ekberg, P., Stigge, M., and Yi, W., 'Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems,' in 'Proceedings of the 32nd International Real-Time Systems Symposium (RTSS),' IEEE, 2011 pp. 13–23.

Guo, Z., *Real-time scheduling of mixed-critical workloads upon platforms with uncertainties*, Ph.D. thesis, The University of North Carolina at Chapel Hill, 2016.

Guo, Z. and Baruah, S., *Mixed-Criticality Real-Time Systems*, pp. 1–20, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-54477-4, 2018, doi:10.1007/978-3-642-54477-4_6-1.

Guo, Z., Santinelli, L., and Yang, K., 'EDF schedulability analysis on mixed-criticality systems with permitted failure probability,' in 'Proceedings - IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2015,' ISBN 9781467378550, 2015 doi:10.1109/RTCSA.2015.8.

Guo et al., S. V. S. A. S. K. D. H. X., K. Yang, 'Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate,' 2018, accepted in the proceedings of the 39th IEEE Real-Time Systems Symposium (RTSS).

Hamdaoui, M. and Ramanathan, P., 'A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines,' IEEE Transactions on Computers, 1995, **44**(12), pp. 1443–1451, doi:10.1109/12.477249.

Hansen, J., Hissam, S., and Moreno, G., 'Statistical-based wcet estimation and validation,' in 'Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET'09),' 2009 .

Hansen, J., Lehoczky, J., Zhu, H., and Rajkumar, R., 'Quantized EDF scheduling in a stochastic enviroment,' in 'Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS'02),' 2002 .

Hardy, D. and Puaut, I., 'Static probabilistic worst case execution time estimation for architectures with faulty instruction caches,' in 'Proceedings of the 21st International Conference on Real-Time and Networked Systems (RTNS'13),' 2013 .

Irving, R. W., 'Np-completeness of a family of graph-colouring problems,' Discrete Applied Mathematics, 1983, **5**(1), pp. 111–117.

Jan, M., Zaourar, L., and Pitel, M., 'Maximizing the execution rate of low-criticality tasks in mixed criticality systems,' in 'Proceedings of the 34th IEEE Real-Time Systems Symposium,' 2013 .

Korte, B. and Vygen, J., *Bin-Packing*, pp. 471–488, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-24488-9, 2012, doi:10.1007/978-3-642-24488-9_18.

Lee, J., Phan, K.-M., Gu, X., Lee, J., Easwaran, A., Shin, I., and Lee, I., 'Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors,' in 'Proceedings of the 35th IEEE Real-Time Systems Symposium (RTSS),' IEEE, 2014 pp. 41–52.

Lehoczky, J., 'Real-time queueing theory,' in 'Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96),' 1996 .

Li, H., *Scheduling mixed-criticality real-time systems*, Ph.D. thesis, The University of North Carolina at Chapel Hill, 2013.

Li, H. and Baruah, S., 'An algorithm for scheduling certifiable mixed-criticality sporadic task systems,' in 'Proceedings of the 31st International Real-Time Systems Symposium (RTSS),' IEEE, 2010 pp. 183–192.

Li, H. and Baruah, S., 'Global mixed-criticality scheduling on multiprocessors,' in 'Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS),' 2012 pp. 166–175.

Liu, D., Spasic, J., Guan, N., Chen, G., Liu, S., Stefanov, T., and Yi, W., 'EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees,' in 'Proceedings of the 37th IEEE Real-Time Systems Symposium,' ISBN 978-1-5090-5303-2, 2016 doi:10.1109/RTSS.2016.013.

López, J. M., Díaz, J. L., and García, D. F., 'Utilization bounds for edf scheduling on real-time multiprocessor systems,' Real-Time Systems, 2004, **28**(1), pp. 39–68.

Majumdar, R., Saha, I., and Zamani, M., 'Performance-aware scheduler synthesis for control systems,' in 'Proceedings of the 9th ACM International Conference on Embedded Software,' ISBN 9781450307147, 2011 doi:10.1145/2038642.2038689.

Manolache, S., Eles, P., and Peng, Z., 'Schedulability analysis of applications with stochastic task execution times,' ACM Trans. Embedded Computing Systems (TECS), 2004, **3**(4), pp. 706–735.

Masrur, A., Müller, D., and Werner, M., 'Bi-level deadline scaling for admission control in mixed-criticality systems,' in 'Proceedings of the 21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA),' IEEE, 2015 pp. 100–109.

Maxim, D. and Cucu-Grosjean, L., 'Response time analysis for fixed-priority tasks with multiple probabilistic parameters,' in 'Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS'13),' 2013 .

Niz, D., Lakshmanan, K., and Rajkumar, R., 'On the scheduling of mixed-criticality real-time task sets,' in 'Proceedings of the 30th International Real-Time Systems Symposium (RTSS),' IEEE, 2009 pp. 291–300.

Pathan, R., 'Schedulability analysis of mixed-criticality systems on multiprocessors,' in 'Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS),' 2012 pp. 309–320.

Rodriguez, P., George, L., Abdeddaım, Y., and Goossens, J., 'Multicriteria evaluation of partitioned EDF-VD for mixed-criticality systems upon identical processors,' in 'Proceedings of the Workshop on Mixed Criticality Systems (WMC),' 2013 .

S. Baruah and A. Burns, 'Towards a more practical model for mixed criticality systems,' in 'Proceedings of the Workshop on Mixed-Criticality Systems (WMC),' 2014 .

Saha, I., Baruah, S., and Majumdar, R., 'Dynamic Scheduling for Networked Control Systems,' in 'Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control,' 2015 doi:10.1145/2728606.2728636.

Santy, F. F., George, L., Thierry, P., and Goossens, J. J., 'Relaxing Mixed-Criticality Scheduling Strictness for Task Sets Scheduled with FP,' in 'Proceedings of the 24th Euromicro Conference on Real-Time Systems,' ISBN 978-1-4673-2032-0, 2012 doi:10.1109/ECRTS.2012.39.

Slijepcevic, M., Kosmidis, L., Abella, J., Nones, E. Q., and Cazorla, F. J., 'Dtm: Degraded test mode for fault-aware probabilistic timing analysis,' in 'Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS'13),' 2013 .

Su, H. and Zhu, D., 'An Elastic Mixed-Criticality Task Model and Its Scheduling Algorithm,' in 'Proceedings of the Design, Automation & Test in Europe Conference & Exhibition,' ISBN 9781467350716, 2013 doi:10.7873/DATE.2013.043.

Tia, T., Deng, Z., Storch, M., Sun, J., Wu, L., and Liu, J., 'Probabilistic performance guarantee for real-time tasks with varying computation times,' in 'Proceedings of 1st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'95),' 1995 .

Vestal, S., 'Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,' in 'Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS),' 2007 .

Welsh, D. J. and Powell, M. B., 'An upper bound for the chromatic number of a graph and its application to timetabling problems,' The Computer Journal, 1967, **10**(1), pp. 85–86.

Xu, H. and Burns, A., 'Semi-partitioned model for dual-core mixed criticality system,' in 'Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS),' ACM, 2015 pp. 257–266.

Zhu, H., Hansen, J., Lehoczky, J., and Rajkumar, R., 'Optimal partitioning for quantized EDF scheduling,' in 'Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02),' 2002 .

VITA

The author, Samsil Arefin, was born in Khulna, Bangladesh. Since his childhood, he developed a passion for technology and eventually completed his bachelor's degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology. After his bachelor's degree, he worked as a software engineer in multiple top companies in Bangladesh, working in cutting-edge technologies. In 2017, he enrolled in Missouri University of Science and Technology to pursue his master's degree. During his time at Missouri S&T, the author worked in the Real-Time Systems research group as a Graduate Research Assistant under the supervision of Dr. Zhishan Guo.

In partial fulfillment of the requirements for the Master of Science in Computer Science from Missouri University of Science and Technology, this thesis is the culmination of that degree. The author received his Master of Science in Computer Science from Missouri S&T in December 2018.