SCHEDULING MIXED-CRITICALITY REAL-TIME SYSTEMS

Haohan Li

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2013

Approved by:

Sanjoy K. Baruah

James H. Anderson

Kevin Jeffay

Montek Singh

Leen Stougie

ABSTRACT

HAOHAN LI: Scheduling Mixed-Criticality Real-Time Systems
(Under the direction of Dr. Sanjoy K. Baruah)

This dissertation addresses the following question to the design of scheduling policies and resource allocation mechanisms in contemporary embedded systems that are implemented on integrated computing platforms: in a multitasking system where it is hard to estimate a task's worst-case execution time, how do we assign task priorities so that 1) the safety-critical tasks are asserted to be completed within a specified length of time, and 2) the non-critical tasks are also guaranteed to be completed within a predictable length of time if no task is actually consuming time at the worst case?

This dissertation tries to answer this question based on the mixed-criticality real-time system model, which defines multiple worst-case execution scenarios, and demands a scheduling policy to provide provable timing guarantees to each level of critical tasks with respect to each type of scenario. Two scheduling algorithms are proposed to serve this model. The OCBP algorithm is aimed at discrete one-shot tasks with an arbitrary number of criticality levels. The EDF-VD algorithm is aimed at recurrent tasks with two criticality levels (safety-critical and non-critical). Both algorithms are proved to optimally minimize the percentage of computational resource waste within two criticality levels. More in-depth investigations to the relationship among the computational resource requirement of different criticality levels are also provided for both algorithms.

Dedicated to my fiancée and my parents.

ACKNOWLEDGEMENTS

I have always been feeling lucky when I pursue the doctoral degree at The University of North Carolina at Chapel Hill, because I have received too much help and support to repay. The first and the most important of all, the best part of my graduate student life is to have Sanjoy Baruah as my advisor. He shows me how attractive the computer science research is, how kind a teacher can be, and how wonderful an academic life will be. It wouldn't be possible for me to become who I am without him, and it is my dream to become who he is.

I would like to express my sincerest appreciation to my committee members: Jim Anderson, Kevin Jeffay, Montek Singh, and Leen Stougie, for the time and energy they spend on my dissertation. It is my great fortune to receive guidance and advices from them during my work. I am also very grateful to my co-authors: Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Nicole Megow, Suzanne Van Der Ster, Bipasa Chattopadhyay, and Insik Shin. It is my honor and my pleasure to work with so many great minds on numerous interesting problems.

I want to thank all my friends in the real-time systems group, from whom I learn a lot: Cong Liu, Mac Mollison, Glenn Elliott, Zhishan Guo, Jeremy Erickson, Bryan Ward, Alex Mills, Andrea Bastoni, Hennadiy Leontyev and Björn Brandenburg. The delightful memory in this great group will never fade away.

I am extremely proud to spend five years in The Department of Computer Science at UNC. I am grateful to the department for awarding me the Computer Science Alumni Fellowship that supports my dissertation writing. Also, I appreciate the infinite help from all the faculty members and staffs. Especially, I would like to say thanks to Jodie Turnbull for her splendid service and unreserved help during my study and job hunting.

I wish to say "thank you" to my parents, for their constant and unconditional love.

Finally, I am deeply thankful to my fiancée Zhongwan for her love, support, trust, passion, and inspiration. Thank you for meeting me; thank you for loving me; thank you for saying "yes". I wish to have my *happily ever after* with you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CA	Certification Authority
CM	Criticality-Monotonic
EDF	Earliest-Deadline-First
EDF-VD	Earliest-Deadline-First with Virtual Deadlines
FAA	Federal Aviation Administration
LCM	Least Common Multiple
LHS	Left-Hand Side
MC	Mixed-Criticality
NP	Non-deterministic Polynomial-time
OCBP	Own-Criticality-Based-Priority
RTCA	Radio Technical Commission for Aeronautics
SWaP	Size, Weight and Power
UAV	Unmanned Aerial Vehicle
WCET	Worst-Case Execution Time
WCR	Worst-Case Reservation

CHAPTER 1

Introduction

Traditional real-time scheduling theory faces challenges in modern computation-intensive and time-sensitive cyber-physical embedded systems. Nowadays, real-time embedded computing systems are widely used in safety-critical environments such as avionics and automobiles. There are two conflicting trends in the development of these systems. One is that the safety assurance requirements are increasingly emphasized. Some critical real-time tasks must never fail to meet their deadlines, even under extremely harsh circumstances. The other is that more functionalities are implemented on integrated platforms due to size, weight and power (SWaP) constraints. Therefore, many non-critical real-time tasks will share and compete for the computational resources with critical tasks. Unfortunately, traditional real-time scheduling theory cannot provide a balance between these two requirements. The existing techniques have to reserve unreasonably large amounts of computational resources to ensure that every real-time task performs correctly under harsh circumstances — even the non-critical ones. This inefficiency makes it desirable that the assumptions, abstractions and objectives in traditional real-time scheduling theory be reconsidered, such that these safety-critical systems will sacrifice neither reliability nor efficiency.

1.1 Overview of Real-Time Systems

Modern embedded systems broadly interact with physical environments, and commonly require that every input signal is responded to within a predictable length of time. In these systems, there are two notions of correctness, *logical* correctness and *temporal* correctness. Logical correctness usually means "to generate the correct results", which is quite commonly required in general computing systems; temporal correctness usually means "to perform

actions at the required time", which is an additional main objective in real-time systems. *Real-time systems* are defined as systems that provide temporal correctness. In real-time systems, the temporal predictability, which is often in the form of guaranteeing every task's response within strict deadlines, is as important as the performance (how fast an individual task can complete) or the throughput (how many tasks can be completed over a long period of time).

In real-time scheduling theory research, the scheduling algorithms that switch tasks and allocate resources in real-time systems are studied. These algorithms are constructed based on real-time task models. These models will be introduced in Section 1.3. These models extract the essential information of the temporal behaviors of the tasks in a real-time system. The scheduling algorithms must predictably assure *a priori* that all tasks are completed by their deadlines, assuming that the tasks follow the specifications in the workload model. Because these guarantees must be analytically proved before the actual execution of the system, there are usually two types of algorithms on scheduling:

- *Scheduling policies*, sometimes called *schedulers*, are the algorithms that control the run-time schedule. The scheduling policies will be executed along with real-time tasks, and make scheduling decisions based on the time and/or the temporal behavior of real-time tasks. Scheduling policies are generally required to be simple and fast because they compete with real-time tasks and occupy computational resources.

- *Schedulability tests* are the algorithms that check *before run-time* if the deadlines are guaranteed to be met. Schedulability tests can be complicated and time-consuming if they can bring in better run-time performance and computational resource efficiency. It is non-trivial to design schedulability tests, especially for real-time tasks with a large variance of run-time behaviors because the tests must guarantee that no deadline is missed in all possible system runs.

This dissertation focuses on a new real-time task model, *the mixed-criticality task model*. Section 1.3 will describe the traditional model, the new model, and their differences. The following chapters in this dissertation will introduce several scheduling algorithms aimed at

the mixed-criticality system model, while Section 1.5 gives an overview of all these scheduling algorithms.

1.2 Motivation

The research of scheduling mixed-criticality systems starts from abstracting a realistic problem — the certification requirement. Many safety-critical embedded systems must pass certain safety certifications. In the certification processes, the certification authorities, such as Federal Aviation Administration (FAA), will verify the safety standards within the system, including the real-time constraints of the safety-critical tasks. It is important to note that the certification authorities tend to be very conservative in the certification. They require that the correctness be demonstrated under extremely rigorous and pessimistic assumptions, which are very unlikely to occur in reality.

Traditional real-time scheduling techniques commonly do not work efficiently on these certifiable systems. The reason is that the scheduling theory is based on abstract task models. In these models, the tasks are usually specified by several parameters: the worst-case execution time, the deadline and the release pattern. The objective is a scheduling policy with a strong requirement: *all deadlines must always be met*. In order to fulfill this requirement, the worst-case execution time (WCET) of a task, which is a parameter that must be determined beforehand, is required never to be exceeded by the actual execution time of this task. In practice, determining an exact WCET value for a task is very difficult and remains an active area of research. Therefore, the WCET parameter used by the certification authorities is typically a very conservative upper bound that highly exceeds the true WCET. Moreover, in typical real-time system models, no isolation exists between tasks in a traditional real-time system because all tasks are treated as equally important. This implies a task will possibly miss its deadline if another task fails to be bounded by its own WCET. As a result, in order to prevent *any* potential deadline miss, very pessimistic WCET values must be used for *all* tasks in certifications. This will inevitably cause severe computational resource waste. Past scheduling techniques that focus on meeting all deadlines are not able to eliminate this waste.

Knowing the shortcomings of the traditional models, how do we abstract a certifiable real-time system, and what kind of scheduling policies do we seek then? The key idea is to observe that in many applications, the consequence of a deadline miss varies among tasks. For example, in the RTCA DO-178B avionics software standard, as listed in Table 1.1, the tasks are divided into five assurance levels, from level A to level E. In the standard, a failure of a level-A task will have catastrophic results (e.g. causing a crash), while a failure of a level-E task will have no influence on flight safety. Under these circumstances, it is reasonable not to presuppose an objective that all low-criticality deadlines are always met. Therefore, the mixed-criticality real-time system model is proposed by Vestal (Vestal, 2007) on the basis of a new assumption that *only high-criticality deadlines are guaranteed to be met if high WCET estimations are used, and all deadlines are guaranteed to be met if low WCET estimations are used*. Under this new assumption, only high-criticality tasks will reserve a large amount of time while several thresholds of possible execution time are also defined. Low-criticality tasks will be executed only if the execution of high-criticality tasks execute shorter than a certain threshold. Now when the certification authorities assume high WCET estimations, the high-criticality tasks will perform correctly; but the system is still able to perform many real-time functionalities if these high-criticality tasks execute normally.

1.3 Models for Real-Time Systems and Mixed-Criticality Systems

In this section, we introduce the models used in real-time systems and mixed-criticality real-time systems. The models in classic real-time systems will be introduced briefly, while detailed examples and formalized definitions (Vestal, 2007; Baruah et al., 2010b) will be given pertaining to models in mixed-criticality real-time systems.

1.3.1 Real-time Jobs and Recurrent Tasks

There are many real-time task models in classic real-time systems, although the principle remains the same: a piece of code becomes available for execution at a time moment in the

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

Table 1.1: DO-178B is a software development process standard, *Software Considerations in Airborne Systems and Equipment Certification*, published by RTCA, Inc. The United States Federal Aviation Administration (FAA) accepts the use of DO-178B as a means of certifying software in avionics applications. RTCA DO-178B assigns criticality levels to tasks categorized by effects on commercial aircraft.

system, takes a certain amount of time to finish its execution, and is required to finish by a given time moment. The variety of the task models is derived from the definition of the manner which the available time, execution time and deadline follow. In this dissertation, we only consider two kind of real-time task models:

- *Real-time jobs.* This is the simplest task model. In this model, a *job*, representing a piece of code, is available at a specified time, and is required to be finished by another specified time. These two time instants are known as *the release time* and *the deadline*. They can be expressed in absolute ("wall-clock") time, or relative time with respect to a given time instant that is defined as time 0 (usually this time instant is when the whole system starts working, or the first release time in the system). In either case, neither of the two time instants will change as time goes on.

- *Real-time sporadic tasks.* This is the most common *recurrent* task model. A *recurrent* task model uses a finite representation to describe a system that may execute for an indefinite length of time. A *sporadic task* represents a piece of code that must be executed repeatedly. Every time when this piece of code is executed, it is treated as a new job. Thus a sporadic task *releases a job* when this piece of code becomes available

to be (repeatedly) executed. The task will have a parameter, its *relative deadline*, so that when a job is released, it will have its deadline as its release time plus the relative deadline. The jobs cannot be released infinitely frequently — a *minimum* inter-arrival time between any two consecutive job releases is specified, and defined as the task's *period*. In a sporadic task system, it is not possible to know the exact release times of the jobs in this system. However, in any time interval that is no longer than a task's period, there can be at most one job release.

In both task models, it is very important to pre-evaluate the amount of time that a job requires, in order to assure that no job will miss its deadline. This amount of time is represented by the job's *worst-case execution time (WCET)*. In this dissertation, we will not discuss the techniques that are used to evaluate a job's execution time. We only assume that this parameter is known for every job, and a job is guaranteed to be completed if it has been accumulatively executing for its worst-case execution time.

As a summary, a real-time job is specified by three parameters: its release time, its deadline, and its worst-case execution time; a real-time sporadic task is also specified by three parameters: its period, its relative deadline, and its worst-case execution time. A real-time sporadic task can generate infinitely many real-time jobs.

Now we can define the system using the previously described models. We will consider the systems that consist of only real-time jobs, or only real-time tasks. To fully describe the properties of a system, we need more terms, which are provided below.

In this dissertation, only *preemptive* systems are considered. *Preemptive* means that at any time, the scheduling policy can suspend the current executing job, and choose another job (that can be executed) to execute. Though preemption causes context and state saving and costs additional time in reality, we assume in this dissertation that any additional time cost has been bounded by the worst-case execution time. Therefore, in our scheduling polices, we will not analytically limit the number of preemptions (pragmatic limitations may be applied, however).

All our previous statements assume *hard real-time systems*, which means that deadlines can never be missed, or the scheduling policy will be determined as faulty. *Soft real-time*

systems, which tolerate deadline miss in certain pre-defined manners, will not be discussed in this dissertation.

The demand of computational resource is an important property of a system. *Load* can be used to describe both real-time jobs and real-time recurrent tasks. It denotes the *maximum fraction* of processor time demand of a system over any time interval. *Utilization* is used only to describe real-time recurrent tasks. It denotes the *overall fraction* of processor time demand of a system. Here the *time demand* over a given interval means the summation of the WCETs of the jobs that are released in this interval and is required do be finished in this interval. The formal definitions can be found in Subsection 1.3.3 and 1.3.4.

1.3.2 Overview of Mixed-Criticality Systems

In this subsection, we introduce the detailed mixed-criticality system model by considering first an example from the domain of unmanned aerial vehicles (UAVs), used for defense reconnaissance and surveillance. The functionalities on board such UAVs may be classified into two levels of criticality:

- Level 1: *mission-critical* functionalities, concerning reconnaissance and surveillance objectives, like capturing images from the ground, transmitting these images to a base station, etc.
- Level 2: *flight-critical* functionalities: to be performed by the aircraft to ensure its safe operation.

For permission to operate such UAVs over civilian airspace (e.g., for border surveillance), it is mandatory that its flight-critical functionalities be certified correct by civilian Certification Authorities (CAs) such as the US Federal Aviation Administration (FAA), which tend to be very conservative concerning the safety requirements. However, these CAs are not concerned with the mission-critical functionalities: these must be validated separately by the system designers (and presumably the customers — those who will purchase the aircraft). The latter are also interested in ensuring the correctness of the flight-critical functionalities, but the notion of correctness adopted in validating these functionalities is typically less rigorous than the one used by the civilian CA's.

This difference in correctness criteria may be expressed by different *Worst-Case Execution Times* (WCET) estimates for the execution of a piece of real-time code. In fact, the CA and the system designers (and other parties responsible for validating the mission-critical functionalities) will each have their own tools, rules, etc., for estimating WCET; the value so obtained by the CA is likely to be larger (more pessimistic) than the one obtained by the system designer. We illustrate via a (contrived) example.

Example 1.1. Consider a system comprised of two jobs: J_1 is flight-critical while J_2 has lower mission-critical criticality. Both jobs arrive at time-instant 0, and have their deadlines at time-instant 10. For $i \in \{1, 2\}$, let $C_i(1)$ denote the WCET estimate of job J_i as made by the system designer, and $C_i(2)$ the WCET estimate of job J_i as made by the CA.

As we have stated above, WCET values determined by the CA tend to be larger than those determined by the system designer. Suppose that $C_1(1) = 3$, $C_1(2) = 5$ and $C_2(1) = C_2(2) = 6$. Consider the schedule that first executes J_1 and then J_2.

- The CA responsible for safety-critical certification would determine that J_1 completes latest by time-instant 5 and meets its deadline. (Note that if the execution time of J_1 is 5 then in the worst case it is not possible to complete J_2 by its deadline; however, this CA is not interested in J_2; hence the system passes certification.)

- The system designers (and other parties responsible for validating the correctness of the mission-critical functionalities) determine that J_1 completes latest by time-instant 3, and J_2 by time-instant 9. Since both jobs complete by their deadlines, the system is determined to be correct by its designers.

We thus see that the system is deemed as being correct by both the CA and the designers, despite the fact that the sum of the WCET's of the jobs at their own criticality levels (6 and 5) exceeds the length of the time window over which they are to execute. □

Current practice in safety-critical embedded systems design for certifiability is centered around the technique of "space-time partitioning". Loosely speaking, *space partitioning* means that each application is granted exclusive access to some of the physical resources on board the platform, and *time partitioning* means that the time-line is divided into slots with each slot being granted exclusively to some (pre-specified) application. Interactions

among the partitioned applications may only occur through a severely limited collection of carefully-designed library routines. This is one of several *reservation-based* approaches, in which a certain amount of the capacity of the shared platform is reserved for each application, that have been considered for designing certifiable mixed-criticality systems. It is known that reservation-based approaches tend to be pessimistic (in the sense of under-utilizing platform resource); for instance, a reservation-based approach to the example above would require that 5 units of execution be reserved for job J_1, and 6 units for job J_2, over the interval $[0, 10)$.

1.3.3 Mixed-Criticality Jobs

Although the example that we considered in Section 1.3.2 is characterized by just two criticality levels, systems may in general have more criticality levels defined. (For instance, the RTCA DO178-B standard in Table 1.1, widely used in the aviation industry, specifies five different criticality levels, with the system designer expected to assign one of these criticality levels to each job. The ISO 26262 standard, used in the automotive domain, specifies four criticality levels, known in the standard as "safety integrity levels" or SILs.)

Accordingly, the formal model that we use allows for the specification of arbitrarily many criticality levels. Let $L \in \mathbb{N}^+$ denote the number of distinct criticality levels in the mixed-criticality system being modeled.

Definition 1.1. A *mixed-criticality job* in the mixed-criticality system is characterized by a 4-tuple of parameters: $J_j = (a_j, d_j, \chi_j, C_j)$, where

- $a_j \in \mathbb{Q}_+$ is the release time;
- $d_j \in \mathbb{Q}_+$ is the deadline, $d_j \geq a_j$;
- $\chi_j \in \mathbb{N}_+$ is the criticality of the job;[1]
- $C_j \in \mathbb{Q}_+^L$ is a vector, the k-th coordinate of which specifies the worst-case execution time (WCET) estimate of job J_j at criticality level k. In a job-specification we will usually represent it by $(C_j(1), \ldots, C_j(L))$. □

[1] If there are only two criticality levels in the system, we can also use level LO and level HI instead of level 1 and 2 when representing χ_j, and denote this system as a *dual-criticality* system.

We will, for the most part, assume that $C_j(k)$ is monotonically non-decreasing with increasing k. This is a reasonable assumption: these $C_j(k)$ values represent *upper bounds*, at different degrees of confidence, on the WCET of the job. Larger values of k correspond to greater degrees of confidence, and are therefore likely to be larger. At any moment, we call a job *available* if its release time has passed and the job has not yet completed execution.

An *instance* I of the MC-schedulability problem consists of a set of n jobs. In this dissertation we assume that there is only one machine (processor) to execute the jobs. However, we have some results on multiprocessor, which is briefly introduced in Section 6.2.

We assume that this processor is *preemptive*: executing jobs may have their execution interrupted at any instant in time and resumed later, with no additional cost or penalty.

To define MC-schedulability we define the notion of a *scenario*.

Definition 1.2. Each job J_j requires an amount of execution time c_j within its *time window* $[a_j, d_j]$. The value of c_j is not known from the specification of J_j, but is only discovered by actually executing the job until it *signals* that it has completed execution. This characterizes the uncertainty of the problem. We call a collection of realized values $(c_1, c_2, \ldots, c_n)$ a *scenario* of instance I. □

Definition 1.3. The *criticality level*, or simply criticality, of a scenario $(c_1, c_2, \ldots, c_n)$ of I is the smallest integer k such that $c_j \leq C_j(k)$ for all $j = 1, \ldots, n$. (If there is no such k, we define that scenario to be *erroneous*.) □

Definition 1.4. A schedule for a scenario $(c_1, \ldots, c_n)$ of criticality k is *feasible* if every job J_j with $\chi_j \geq k$ receives execution time c_j during its time window $[a_j, d_j]$. □

A *clairvoyant* scheduling policy knows the scenario of I, i.e., $(c_1, \ldots, c_n)$, prior to determining a schedule for I.

Definition 1.5. An instance I is *clairvoyantly-schedulable* if for each non-erroneous scenario of I there exists a feasible schedule. □

In contrast to clairvoyant scheduling policies, an *on-line* scheduling policy discovers the value of c_j only by executing J_j until it signals completion. In particular, the criticality level of the scenario becomes known only by executing jobs. At each time instant, scheduling decisions can be based only on the partial information revealed thus far.

Definition 1.6. An on-line scheduling policy is *correct* for instance I if for any non-erroneous scenario of instance I the policy generates a feasible schedule. □

Definition 1.7. An instance I is *MC-schedulable* if there exists a correct on-line scheduling policy for instance I. □

It is very obvious to see that a MC-schedulable instance I must be also clairvoyant-schedulable because otherwise there will be scenarios that do not have a feasible schedule.

The **MC-schedulability** problem is to determine whether a given instance I is MC-schedulable or not.

Example 1.2. Consider an instance I of a *dual-criticality* system: a system with $L = 2$. I is comprised of 2 jobs: job J_1 has criticality level 1 (which is the lower criticality level), and the other job has the higher criticality level 2.

$J_1 = (0, 2, 1, (1, 1))$

$J_2 = (0, 3, 2, (1, 3))$

For this example instance, any scenario in which c_1 and c_2, are no larger than 1, has criticality 1; while any scenario not of criticality 1 in which c_1 and c_2 are no larger than 1, and 3, respectively, has criticality 2. All remaining scenarios are, by definition, erroneous. It is easy to verify that this instance is clairvoyantly-schedulable.

Policy S_0, described below, is an example of an on-line scheduling policy for instance I:

S_0**:** Execute J_2 over [0,1]. If J_2 has no remaining execution (i.e., c_2 is revealed to be no greater than 1), then continue with scheduling J_1 over $(1, 2]$; else continue by completely scheduling J_2.

It is easy to see that policy S_0 is correct for instance I. However, S_0 is not correct if we modify the deadline of J_1 obtaining the following instance I':

$J_1 = (0, 1, 1, (1, 1))$

$J_2 = (0, 3, 2, (1, 3))$

After the modification, S_0 will cause J_1 to miss its deadline if J_2 has no remaining execution at time 1.

It is easy to see that I' is clairvoyantly schedulable but not MC-schedulable. Any on-line scheduling policy that starts with executing J_1 will cause J_2 to miss its deadline if c_2 is revealed to be 3; any on-line scheduling policy that starts with executing J_2 will cause J_1 to miss its deadline if c_2 is revealed to be no greater than 1. □

Speedup factors are a useful conceptual characterization of the effectiveness of a scheduling policy, and may provide valuable insight into the policy's properties.

Definition 1.8. The speedup factor x for a scheduling policy A is defined as the minimum factor by which the speed of the processor would need to be increased such that all instances/systems that are schedulable according to a clairvoyant scheduling policy on a processor become schedulable under the policy A.

A speedup factor x of the scheduling policy A is called *exact* if there exists an instance/system that is schedulable on processor(s) of speed 1 by an optimal (possibly clairvoyant) scheduling policy but is not schedulable by A on any processor(s) of speed lower than x.

A scheduling policy A with speedup factor x is called *optimal with respect to speedup factors* if there exists an instance/system that is schedulable on processor(s) of speed 1 by the optimal (and *clairvoyant*) scheduling policy but is not schedulable by *any on-line* scheduling policies on any processor(s) of speed lower than x. □

Loads are also a useful conceptual characterization of the effectiveness of a scheduling policy, and provide more in-depth investigation to the relationship among the computational resource requirement of different criticality levels. Analogous to the load concept in traditional real-time task models, we find it convenient to define loads of a MC instance I in different criticality levels:

Definition 1.9. Given a MC instance I, the load of I in criticality level $k(1 \leq k \leq L)$ is defined as the maximum ratio between the sum of criticality level k WCETs in any time interval and the length of this time interval. It can be written formally as:

$$\ell_k = \max_{0 \leq t_1 < t_2} \frac{\sum_{J_i \,:\, \chi_i \geq k \wedge t_1 \leq a_i \wedge d_i \leq t_2} C_i(k)}{t_2 - t_1} \tag{1.1}$$

□

Informally, ℓ_k is the largest proportion of processor occupation that the system expects to have to deal with during run-time while executing instance I in a behavior of criticality level k. Clearly, it is necessary (albeit not sufficient) that all ℓ_k be no larger than the speed of the processor on which I is to be executed.

All ℓ_k for a MC instance I with n jobs can be determined in time that is polynomial in n. To see this, we observe that only such values of t_1 and t_2 need be considered where t_1 is equal to some a_i and t_2 is equal to some d_i. There are no more than n^2 possible such $[t_1, t_2)$ intervals, and computing the sum of the WCET estimates over each interval takes $O(n)$ time. Thus even with the brute-force method, we can compute both loads in $O(n^3)$ time.

1.3.4 Mixed-Criticality Recurrent Tasks

Definition 1.10. A mixed-criticality sporadic task in the mixed-criticality system is characterized by a 4-tuple of parameters: $\tau_j = (\chi_j, C_j, T_j, D_j)$, where

- $\chi_j \in \mathbb{N}_+$ is the criticality of the task;
- $C_j \in \mathbb{Q}_+^L$ is a vector, the k-th coordinate of which specifies the worst-case execution time (WCET) estimate at criticality level k;
- $T_j \in \mathbb{Q}_+$ is the period;
- $D_j \in \mathbb{Q}_+$ is the relative deadline.

Task τ_j generates a potentially infinite sequence of jobs, with successive jobs being released at least T_j time units apart. Each such job has a deadline that is D_j time units after its release. The criticality of each such job is χ_j, and it has the WCET estimation vector as $C_j = (C_j(1), \ldots, C_j(L))$. □

A MC *sporadic task set* is specified by specifying a finite number of such sporadic tasks. As with traditional (non-MC) systems, such a MC sporadic task set can potentially generate infinitely many different MC instances (collections of jobs), each instance being obtained by taking the union of one sequence of jobs generated by each sporadic task.

A MC *implicit-deadline sporadic task set* is a MC sporadic task set with $T_k = D_k$ for all τ_k. A MC *arbitrary-deadline sporadic task set* is a MC sporadic task set in which no

restriction is placed on the relation between periods and deadlines. Implicit-deadline sporadic tasks are a special case of arbitrary-deadline sporadic tasks.

The definition of loads extends in the obvious manner to systems of sporadic tasks: The load ℓ_k of sporadic task system τ is defined to be the largest value that ℓ_k can have, for any instance I generated by τ. However, the computation of loads in sporadic systems is harder than in in job sets (ususally impossible in polynomial time). We will introduce the computation of loads in sporadic systems later.

Utilization is another conceptual characterization of a sporadic task system.

Definition 1.11. Given a MC sporadic task set τ, the utilization of τ in criticality level k $(1 \leq k \leq L)$ is defined as

$$U_k = \sum_{\tau_j : \chi_j \geq k} \frac{C_j(k)}{T_j}. \tag{1.2}$$

□

Moreover, the utilization at level k of jobs that are of criticality level i is defined as

$$U_k(i) = \sum_{\tau_j : \chi_j = i} \frac{C_j(k)}{T_j}. \tag{1.3}$$

□

Utilizations are defined over all sporadic task sets. but it is the most useful for implicit-deadline sporadic task set, or in soft-real-time systems. In this dissertation, we only use utilizations to evaluate scheduling policies in Chapter 4. It is easy to see that in a MC implicit-deadline sporadic task set, $\ell_k = U_k$ for all $1 \leq k \leq L$. The reason is that in any given time interval $[t_1, t_2)$, the time demand can not exceed $U_k(t_2 - t_1)$. Therefore $U_k \geq \ell_k$ must hold. The "equal" can be reached at the LCM of all periods.

1.4 Thesis Statement

The central thesis explored in our research is that *efficient resource allocation in systems that are subject to multiple different correctness criteria requires the development of new approaches for resource-allocation and scheduling*. This dissertation describes our efforts to date towards developing such approaches. Therefore, the thesis statement is as follows:

New methods can be discovered to schedule real-time systems with multiple criticalities. The methods can supply multiple temporal predictability assertions with respect to multiple WCET specifications. The assertions can be defined and measured through a formalized description. The methods can be efficiently implemented with acceptable computational complexities.

1.5 Contributions

The first and basic question after proposing the model is a purely algorithmic problem: how to schedule mixed-criticality jobs, with specified release-times and deadlines, when no dependencies among these jobs exist? The new assumptions in the mixed-criticality model require new principles. While the traditional real-time system scheduling favors only urgent jobs, mixed-criticality systems must also prioritize high-criticality jobs so that they are prepared for potentially long execution. The compromise between urgency and significance results in exponential choices, and hence leads to the fact that mixed-criticality scheduling is NP-hard in the strong sense (Baruah et al., 2010c, 2012b). In the absence of an efficient optimal algorithm, the first efficient approximation algorithm named *Own-Criticality-Based-Priority* (OCBP) algorithm is proposed to generate a correct priority list for mixed-criticality jobs in polynomial time. This algorithm recursively searches a lowest-priority job by simulating the behavior of all other jobs according to the candidate's own criticality. The performance of this algorithm is quantified in the form of speedup factors. The speedup factor of OCBP algorithm is 1.618 (the golden ratio) for two criticality levels, which is a significant improvement compared with current practice's speedup factor 2, and is proved to be the minimal speedup factor the on-line scheduling policies can reach. The speedup factors for more criticality levels are also calculated, and it is shown that the OCBP algorithm gives an $O(L/\ln L)$ speedup factor to L criticality levels as opposed to the $O(L)$ speedup factor of the current worst-case reservation method. A more in-depth investigation to the relationship among the processor loads of different criticality levels is also made so that we understand the connections among the criticality levels more precisely and quantitatively.

The more general and popular abstraction to real-time tasks in real-time system research is the sporadic task model, where the jobs are released recurrently with minimum time gaps. Thus the second question to the mixed-criticality system is whether the mixed-criticality sporadic tasks can be efficiently scheduled. In order to answer this question, the classic Earliest-Deadline-First (EDF) algorithm is specialized to support mixed-criticality systems. In the traditional real-time scheduling, EDF is proved to be an optimal exact algorithm on preemptive uniprocessor platform. Its philosophy of favoring urgent jobs, in the form of always choosing the job with the earliest deadline to execute, provably maximizes the use of processor capacity in the traditional real-time systems. Yet in order to promote high-criticality tasks in mixed-criticality systems as mentioned above, these tasks will be given earlier virtual deadlines to reflect their importance over low-criticality tasks. This modified EDF algorithm (named EDF-VD, where VD stands for virtual deadlines) has a speedup factor of 4/3 for mixed-criticality sporadic systems with two criticality levels and implicit deadlines, while this factor is also shown as minimal for the mixed-criticality implicit-deadline sporadic task systems. This modified EDF algorithm is also extended to mixed-criticality arbitrary-deadline sporadic task sets, and is proved to have a speedup factor as 1.83.

Besides these two main contributions, other research results will also be mentioned in Chapter 6. The OCBP algorithm is generalized according to the sporadic task model. The challenge is that a sporadic task can release infinite jobs at indefinite time instants. An on-line priority adjustment technique is proposed to deal with unanticipated job releases while preserving the speedup factor. It uses a priority list computed by the OCBP algorithm for a longest and densest possible job release sequence, and recomputes the priority list to dynamically secure the correctness of the ongoing schedule at the instant when a new job releases. The EDF-VD algorithm is extended to the main two categories in the multiprocessor scheduling theory: the *global* multiprocessor scheduling (tasks and jobs can migrate among processors) and the *partitioned* multiprocessor scheduling (tasks are assigned to dedicated processors). An initial effort is made on analyzing the mixed-criticality multiprocessors scheduling problem. Both based-on the EDF-VD algorithm, the global scheduling policy adjusts the deadlines of high-criticality tasks so that the system is schedulable when the

tasks are globally scheduled according to the adjusted deadlines; whereas the partitioned scheduling policy partitions the tasks to subsets so that each subset is schedulable by EDF-VD on a uniprocessor. The speedup factors for these algorithms are respectively 3.24 and 2.67, as opposed to the speedup factor 4 of the naive worst-case reservation method.

CHAPTER 2

Prior Work

In this chapter, we briefly review important research results related to mixed-criticality scheduling. In Section 2.1, important results in traditional real-time scheduling theory that are helpful to mixed-criticality scheduling are introduced, including the optimal scheduling policy EDF, and the relationship between loads, utilizations and schedulability. In Section 2.2, research results on the mixed-criticality model in this dissertation and on other similar real-time task models are presented.

2.1 Real-Time Scheduling Theory

In this section, we first introduce the EDF scheduling policy, then show its optimality and schedulability conditions.

Definition 2.1. Earliest-deadline-first (EDF) scheduling policy always selects the job with the earliest deadline in all available jobs to execute at any time moment. □

Theorem 2.1. (Liu and Layland, 1973; Horn, 1974) *If a real-time job instance or sporadic task set is schedulable on a uniprocessor by any scheduling policy, it is also schedulable by EDF.*

This theorem, usually stated as "EDF is optimal", is proved by transforming the schedule produced by the feasible scheduling policy to an EDF schedule. It can be shown that for any schedule on a specific behavior of the job/task set, if the executions are "swapped" to follow the EDF manner, no deadline will be missed due to the swapping which places the execution of later-deadline jobs after earlier-deadline jobs. The proof requires that the system is preemptive, and furthermore, that the time demand of a job is independent of

the moment and order of its execution. Therefore, this scheduling policy is not optimal any more when applied to mixed-criticality task models, because if the execution order of the jobs is changed, the time demand may differ — low-criticality jobs may be ignored after the high-criticality behavior is revealed.

Example 2.1. Consider an dual-criticality job instance I that consists of two jobs J_1 and J_2. As we denote a job as $J_j = (a_j, d_j, \chi_j, C_j)$, the two jobs are represented as

$$J_1 = (0, 4, 2, (2, 4))$$

$$J_2 = (1, 3, 1, (1, 1)).$$

Now if we consider the EDF scheduling policy in *low-criticality* behavior of this instance, the run-time behavior will be: (1) at time instant 0, start executing J_1; (2) at time instant 1, suspend J_1 and start executing J_2, because J_2 is available to execute and has the earliest deadline in all available jobs (J_1 and J_2); (3) at time instant 2, J_2 is completed so start executing J_2 again; (4) at time instant 3, complete J_1 (because it is the low-criticality behavior) which meets its deadline.

It is important to note that even if we do not know the parameters (including release times, deadlines and WCETs) of all or any jobs, the scheduling decisions made by EDF scheduling policy will remain the same. Like in the example, assuming that J_2's release time isn't known beforehand, at time instant 1 when J_2 is actually released, EDF scheduling policy will pick J_2 for execution.

However, if we consider the *high-criticality* behavior of this instance which includes a criticality change, EDF scheduling policy isn't correct (thus non-optimal) any more. If we simulate the run-time behavior again, it will be: (1) at time instant 0, start executing J_1; (2) at time instant 1, suspend J_1 and start executing J_2; (3) at time instant 2, J_2 is completed so start executing J_2 again; (4) at time instant 5, complete J_1 (because it is the high-criticality behavior) which misses its deadline.

A correct scheduling policy for this example can be a priority-based policy: assign J_1 a higher priority than J_2. Therefore, J_1 will keep executing until time instant 2. If it is completed by then, J_2 will be executed and meet its deadline; otherwise keep executing J_1

and discard J_2, because J_2 can miss its deadline validly if J_1 uses more than $C_1(1) = 2$ time units. □

Theorem 2.2. (Baruah et al., 1990) *A real-time job instance or sporadic task set is schedulable by EDF if and only if its load is no greater than 1.*

The original theorem in (Baruah et al., 1990) is presented in the form of DBF (demand bound function). Basically, it states the fact that as long as the time demand does not exceed the processor capacity in any time interval (or briefly "processor is not over-utilized"), the system is schedulable by EDF.

Though it appears that the calculation of the load of a system is a good method to perform a schedulability test, the following two theorems show that the calculation for sporadic task sets is computationally expensive.

Theorem 2.3. (Baruah et al., 1990; Eisenbrand and Rothvoß, 2010) *The problem of deciding whether a real-time arbitrary-deadline sporadic task set is schedulable is coNP-hard.*

Theorem 2.4. (Baruah et al., 1990) *The load of a real-time arbitrary-deadline sporadic task set can be computed in pseudo-polynomial time if the utilization of the task set is less than 1.*

Theorem 2.3 and 2.4 indicate that there is no polynomial-time schedulability test for arbitrary-deadline sporadic task sets, even though it is known that EDF is the optimal scheduling policy. However, for implicit-deadline sporadic task sets, the schedulability test is much more efficient.

Theorem 2.5. (Liu and Layland, 1973) *A real-time implicit-deadline sporadic task set is schedulable by EDF if and only if its utilization is no greater than 1.*

Theorem 2.5 is discovered much earlier than Theorem 2.2. Actually, Theorem 2.5 is a corollary of Theorem 2.2 because the utilization is equal to the load for an implicit-deadline sporadic task set. Because Theorem 2.5 provides a very efficient schedulability test, it is very widely applied and cited.

The theorems in this section provide the basic expectations to time complexities of schedulability tests for different real-time task models. In Chapter 3, the load-based

schedulability test shows the relationship between schedulability and loads in mixed-criticality jobs, but it is also an efficient schedulability test since the calculation of loads for real-time jobs can be done in polynomial time. In Chapter 4, the utilization-based schedulability test is applied to mixed-criticality implicit-deadline sporadic tasks. In Chapter 5, the load-based schedulability test is applied to mixed-criticality arbitrary-deadline sporadic tasks. In contrast to the results in classic real-time task models, none of these tests in this dissertation keep optimal. The reason why only approximate schedulability tests are pursued is introduced in the following section.

2.2 Mixed-Criticality Scheduling Theory

In this section, we discuss the development of the mixed-criticality task model and scheduling theory, important results in mixed-criticality scheduling theory, and several real-time task models that are similar to the mixed-criticality task model.

The mixed-criticality scheduling problem arises from multiple certification requirements. Vestal presents the concept of applying more conservative worst-case execution time parameters to safety-critical tasks in preemptive uniprocessor recurrent real-time task systems, in order to obtain higher confidence of timing assurance at higher certification level (Vestal, 2007). Also in that paper, the multi-criticality task model is formalized and the fixed-priority response time analysis is conducted. Later in (Baruah and Vestal, 2008), Baruah and Vestal propose a more precise schedulability test in accordance with a hybrid scheduling algorithm that conflates fixed-task-priority scheduling and EDF based on the multi-criticality task model in (Vestal, 2007). The mixed-criticality scheduling model in this dissertation which formalized the *behavior* of tasks and *correctness* of algorithms is presented by Baruah et al. in a more abstracted manner in (Baruah et al., 2010b). The most significant change is that the multi-criticality task model in (Vestal, 2007) and (Baruah and Vestal, 2008) requires that the low-criticality tasks are assigned a larger WCET, and must execute for its WCET in high-criticality behavior; however, in mixed-criticality scheduling model, the run-time scheduling algorithm is permitted to prevent low-criticality tasks from being executed if the algorithm detects high-criticality behavior.

In (Baruah et al., 2010c, 2012b), the MC-schedulability problem in mixed-criticality task model is proven to be NP-hard in the strong sense even in very simple cases.

Theorem 2.6. (Baruah et al., 2010c, 2012b) *MC-schedulability problem is NP-hard in the strong sense, even when all release times are identical and there are only two criticality levels.*

The strong NP-hardness of MC-schedulability problem indicates that neither polynomial nor pseudo-polynomial time algorithms exist to exactly decide whether there is a scheduling policy for a mixed-criticality job instance or task set. As a result, research on scheduling mixed-criticality systems only focuses on seeking efficient *approximation algorithms*. The OCBP (Own-Criticality-Based-Priority) algorithm for mixed-criticality jobs is proposed and analyzed in (Baruah et al., 2010b), (Baruah et al., 2010a), (Baruah et al., 2010c) and (Baruah et al., 2012b). This algorithm will be described in detail in Chapter 3. Park and Kim propose a new algorithm using dynamic programming to schedule dual-criticality jobs in (Park and Kim, 2011), which dominates OCBP algorithm for two criticality levels. Baruah and Fohler propose a time-triggered algorithm to schedule dual-criticality jobs, with the 1.618 speedup factor as well (Baruah and Fohler, 2011). This algorithm is the first *non-work-conserving* mixed-criticality scheduling algorithm (non-work-conserving means that the processor can be idle when there is at least one job available). The connection between the computational resource demands in each criticality level for dual-criticality mixed-criticality jobs scheduled by OCBP algorithm is further investigated in (Li and Baruah, 2010b). This result will be extended to an arbitrary number of criticality levels in Chapter 3.

A more widely-used real-time task model is the *sporadic task model* where the jobs are released recurrently with minimal gaps between adjacent releases, instead of having independently specified release times in the independent job model. The mixed-criticality model is hence extended to *mixed-criticality sporadic task systems*. The OCBP algorithm is enhanced to support sporadic systems in (Li and Baruah, 2010a) by applying the algorithm on a sufficiently long job release sequence to get an initial priority list, and updating the priority list on-line at certain time instants to maintain the correctness of the priority list. Both the off-line schedulability test and run-time updating will cost pseudopolynomial time.

In (Guan et al., 2011), Guan et al. improve this algorithm to a new version named PLRS (Priority-List-Reuse-Scheduling) algorithm, which updates the priority list in $O(n^2)$ time.

The EDF (Earliest-Deadline-First) algorithm is the optimal scheduling policy for classic uniprocessor real-time sporadic task systems, thus it has been modified to support mixed-criticality sporadic task systems in various ways. Baruah et al. propose EDF-VD (EDF with Virtual Deadlines) algorithm that schedules mixed-criticality *implicit-deadline* sporadic task systems (Baruah et al., 2011a). EDF-VD algorithm shrinks the high-criticality deadlines proportionally by a certain factor so that the high-criticality tasks will be promoted by the EDF scheduler. In (Baruah et al., 2011a), an $O(n)$ time sufficient schedulability test with derived speedup factors is proposed while the logarithmic run-time complexity of EDF scheduler is preserved. The speedup factor of EDF-VD algorithm for dual-criticality system is shown as 1.618 in (Baruah et al., 2011a). The speedup factor is improved to 1.33 in (Baruah et al., 2012a). It is also shown that EDF-VD algorithm is optimal with respect to speedup factors in all scheduling policies for dual-criticality implicit-deadline systems in (Baruah et al., 2012a). The analysis and proof of optimal speedup factor will be shown in Chapter 4. Ekberg and Yi propose a new EDF-based algorithm for dual-criticality *constrained-deadline* systems that formulates the demand-bound function in each criticality, assigns every high-criticality task an adjusted relative deadline and schedules the system by EDF according to the adjusted deadlines in (Ekberg and Yi, 2012). The adjusted deadlines are calculated through an off-line procedure which tunes the deadlines in pseudopolynomial time so that the time demand never exceeds the time supply in each criticality.

There is a special class of scheduling policies, *fixed-priority scheduling* where all jobs in one task share the same priority. In (Dorin et al., 2010), Dorin et al. showed that the fixed-priority scheduling of mixed-criticality sporadic task systems with L criticality levels cannot have a speedup factor smaller than L. In (Baruah et al., 2011b), Baruah et al. proposed a fixed-priority scheduling algorithm named AMC (Adaptive Mixed-Criticality) algorithm, which effectively undertakes the response-time analysis and assign priorities to tasks in mixed-criticality constrained-deadline sporadic task systems. Santy et al. discuss the possibility of allowing low-criticality tasks to proceed with their execution even if high-criticality behavior is detected under fixed-priority scheduling in (Santy et al., 2012).

The multiprocessor mixed-criticality scheduling problem has seldom been studied before (Li and Baruah, 2012). In (Mollison et al., 2010), Mollison et al. establish a hierarchical scheduling framework that aims at practically scheduling tasks in multiple criticality levels on a multi-core platform. Herman et al. present the overhead, isolation and synchronization issues when implementing this framework robustly in real-time operating systems in (Herman et al., 2012). However, in (Mollison et al., 2010) and (Herman et al., 2012), the periods of tasks are assumed harmonic and monotonic with respect to criticality levels. In (Baruah et al., 2011a), Baruah et al. show a modified multiprocessor version of OCBP algorithm that schedules dual-criticality independent jobs on identical multiprocessors, and prove the existence of a polynomial-time approximation scheme (PTAS) that partitions dual-criticality sporadic tasks to multiprocessors with EDF-VD scheduler. In (Pathan, 2012), Pathan studies the fixed-priority global and partitioned scheduling on multiprocessors and provides a schedulability test based-on response-time analysis.

Some criticality-related but not certification-cognizant problems are discussed in other papers. De Niz et al. propose another mixed-criticality system model in (de Niz et al., 2009) and (Lakshmanan et al., 2010) where at most one high-criticality task can overrun. Their solution, *zero-slack scheduling* seeks the critical instant after which the high-criticality tasks will have insufficient time to meet their deadlines when overloaded. In (Pellizzoni et al., 2009) and (Yun et al., 2012), Pellizzoni et al. focuses on the the technique of temporal and physical isolation among different criticality levels.

Many other papers discuss problems that are similar to the mixed-criticality scheduling problem. The mode-change protocol research, like in (Real and Crespo, 2004), (Phan et al., 2009) and (Phan et al., 2011), focuses on the response-time analysis and resource allocation techniques when the task set is changed and the pending/partially-processed jobs are required to be completed, transferred or discarded. It deals with a much more general scenario than criticality change in mixed-criticality system, yet does not typically provide specific mixed-criticality scheduling policies. The overloaded real-time scheduling research, like in (Baruah and Haritsa, 1997) and (Koren and Shasha, 2003), tries to maximize the ratio of the completed tasks over all tasks, if possible overloading happens. However, this model is not able to address our problem that different sets of deadlines should be assertively met

on different criticality levels. The bandwidth-preserving servers and compositional analysis research, like in (Ghazalie and Baker, 1995) and (Shin and Lee, 2004), focuses on how to ensure that a task set will not use more than a specified processor capacity budget. Though we have similar requirements in mixed-criticality systems (low-criticality tasks must not overrun), the tools in bandwidth-preserving servers and compositional analysis are aimed at providing much more than our requirement. In mixed-criticality systems, we need to guarantee that no single job will overrun, which can be implemented by simply monitoring a single job's execution time; however, bandwidth-preserving servers and compositional analysis are used to enforce that a collection of jobs/tasks must not use more than a given proportion of the processor capacity, which is too complicated and beyond the necessary functionalities we need.

CHAPTER 3

Scheduling Mixed-Criticality Jobs

In this chapter, we discuss the scheduling of mixed-criticality jobs. In Section 3.2, we briefly introduce a straightforward worst-case reserving solution that simply maps mixed-criticality jobs to a traditional real-time jobs. Then in Section 3.3, 3.4 and 3.5, we introduce our solution, OCBP algorithm, including its detailed description, a load-based schedulability test, and the evaluation of its performance through the speedup factor metric.

3.1 Overview

Since MC-schedulability is intractable even for dual-criticality instances, we concentrate here on sufficient (rather than exact) MC-schedulability conditions that can be verified in polynomial time. We study two widely-used scheduling policies that yield such sufficient conditions and compare their capabilities under the *resource augmentation metric*: the minimum speed of the processor needed for the algorithm to schedule all instances that are MC-schedulable on a unit-speed processor. We show that the second policy we present outperforms the first one in terms of the resource augmentation metric, in the sense that it needs lower-speed processors to ensure such schedulability.

Run-time support for mixed criticality. In scheduling mixed-criticality systems, the kinds of performance guarantees that can be made depend upon the forms of support that are provided by the run-time environment upon which the system is being implemented. A particularly important form of platform support is the ability to *monitor* the execution of individual jobs, i.e., being able to determine how long a particular job has been executing.

Why is such a facility useful? In essence, knowledge regarding how long individual jobs have been executing allows the system to become aware, during run-time, when the

criticality level of the scenario changes from a value k to the next-higher value $k+1$, due to some job executing beyond its level-k WCET without signalling completion; this information can then be used by the run-time scheduling and dispatching algorithm to no longer execute criticality-k jobs once the transition has occurred.

In the remainder of this section, we assume that this facility to monitor the execution of individual jobs is provided by the run-time environment. We may therefore make the assumption that for each job J_j, $C_j(k) = C_j(\chi_j)$ for all $k \geq \chi_j$. That is, no job executes longer than the WCET at its own specified criticality. This is without loss of generality for any correct scheduling policy: any such policy will immediately interrupt (and no longer schedule) a job J_j if its execution time c_j exceeds $C_j(\chi_j)$, since this makes the scenario of higher criticality level than χ_j, and therefore the completion of J_j becomes irrelevant for the scenario.

3.2 Worst-Case Reservation Scheduling

One straightforward approach is to map each MC job J_j into a traditional (i.e., non-MC) job with the same arrival time a_j and deadline d_j and processing time $c_j = C_j(\chi_j) = \max_k C_j(k)$ (by monotonicity), and determine whether the resulting collection of traditional jobs is schedulable using some preemptive single machine scheduling algorithm such as the *Earliest Deadline First* (EDF) rule[1]. This test can clearly be done in polynomial time. We will refer to mixed-criticality instances that are MC-schedulable by this test as *worst-case reservation schedulable* (WCR-schedulable) instances.

Theorem 3.1. *If an instance is WCR-schedulable on a processor, then it is MC-schedulable on the same processor. Conversely, if an instance I with L criticality levels is MC-schedulable on a given processor, then I is WCR-schedulable on a processor that is L times as fast, and this factor is tight.*

[1]In fact, this approach forms the basis of current practice, as formulated in the ARINC-653 standard: each J_j is guaranteed $C_j(\chi_j)$ units of execution in a *time partitioned* schedule, obtained by partitioning the time-line into distinct slots and only permitting particular jobs to execute in each such slot.

Proof. If instance I is WCR-schedulable then for each job the maximum amount of time the job may execute is reserved between its arrival time and its deadline. Hence it is MC-schedulable.

Suppose now that instance I is MC-schedulable. If we were to use a separate processor for each of the L criticality levels, then each job will receive its maximum processing time between arrival time and deadline e.g. by using EDF on the machine corresponding to its criticality level. Hence, by processer sharing, WCR-schedulability on one processor of speed L times faster follows immediately.

Finally, we show that there exist instances with L criticality levels that are MC-schedulable on a given processor, but not WCR-schedulable on a processor that is less than L times as fast.

Consider the instance I comprised of the following L jobs:

$$\begin{aligned} J_1 &= (0, 1, 1, (1, 1, \ldots, 1, 1)) \\ J_2 &= (0, 1, 2, (0, 1, \ldots, 1, 1)) \\ \vdots & \qquad\qquad \vdots \\ J_L &= (0, 1, L, (0, 0, \ldots, 0, 1)) \end{aligned}$$

This instance is MC-schedulable on a unit-speed processor by the scheduling policy of assigning priority in criticality-monotonic (CM) order: $J_L, J_{L-1}, \ldots . J_2, J_1$. Any scenario $(c_1, c_2, \ldots, c_L)$ with $c_h > 0$, $h \geq 2$, and $c_j = 0$ for all $j > h$, has criticality level h, hence all jobs of lower criticality level, in particular J_1, are not obliged to meet their deadline, and job h will meet its deadline. On the other hand, in any scenario of criticality level 1, $c_2 = c_3 = \ldots = c_L = 0$ and $c_1 \in [0, 1]$, hence all jobs meet their deadline.

However, WCR-schedulability requires that each job J_j is executed for $C_j(\chi_j) = 1$, $j = 1, \ldots, L$ before common deadline 1, which clearly can only be achieved on a processor with speed at least L. □

3.3 Own-Criticality-Based-Priority Algorithm

We now consider another schedulability condition, OCBP-schedulability, that offers a performance guarantee (as measured by the processor speedup factor) that is superior to the performance guarantee offered by the WCR-approach. OCBP-schedulability is a constructive test: we determine off-line, before knowing the actual execution times, a total ordering of the jobs in a priority list and for each scenario execute at each moment in time the available job with the highest priority.

The priority list is constructed recursively using the approach commonly referred to in the real-time scheduling literature as the "Audsley approach" (Audsley, 1991, 1993); it is also related to a technique introduced by Lawler (Lawler, 1973). First determine the lowest priority job: Job J_i may be assigned the lowest priority if there is at least $C_i(\chi_i)$ time between its release time and its deadline available when every other job J_j is executed before J_i for $C_j(\chi_i)$ time units (the WCET of job J_j according to the criticality level of job i). This can be determined by simulating the behavior of the schedule under the assumption that every job other than J_i has priority over J_i (and ignoring whether these other jobs meet their deadlines or not — i.e., they may execute under any relative priority ordering, and will continue executing even beyond their deadlines). The procedure is repeatedly applied to the set of jobs excluding the lowest priority job, until all jobs are ordered, or at some iteration a lowest priority job does not exist. If job J_i has higher priority than job J_j we write $J_i \rhd J_j$.

Because the priority of a job is based only on its own criticality level, the instance I is called *Own Criticality Based Priority OCBP)-schedulable* if we find a complete ordering of the jobs.

If at some recursion in the algorithm no lowest priority job exists, we say the instance is not OCBP-schedulable. We can simply argue that this does not mean that the instance is not MC-schedulable: Suppose that scheduling according to the fixed priority list J_1, J_2, J_3 with $\chi_2 = 1$ and $\chi_1 = \chi_3 = 2$, proves the instance to be schedulable. It may not be OCBP-schedulable since this does not take into account that J_2 does not need to be executed at all if J_1 receives execution time $c_1 > C_1(1)$.

It is evident that the OCBP priority list for an instance of n jobs can be determined in time polynomial in n: at most n jobs need be tested to determine whether they can be the lowest-priority job; at most $(n-1)$ jobs whether they can be the 2nd-lowest priority jobs; etc. Therefore, at most $n + (n-1) + \cdots + 3 + 2 + 1 = O(n^2)$ simulations need be run, and each simulation takes polynomial time.

We illustrate the operation of the OCBP priority assignment algorithm by an example:

Example 3.1. Consider the instance comprised of the following three jobs. J_1 is not subject to certification, whereas J_2 and J_3 must be certified correct.

J_i	a_i	d_i	χ_i	$C_i(1)$	$C_i(2)$
J_1	0	4	1	2	2
J_2	0	5	2	2	4
J_3	0	10	2	2	4

Let us determine which, if any, of these jobs could be assigned lowest priority according to the OCBP priority assignment algorithm:

- If J_1 were assigned lowest priority, J_2 and J_3 could consume $C_2(1) + C_3(1) = 2 + 2 = 4$ units of processor capacity over $[0, 4)$, thus leaving no execution for J_1 prior to its deadline.

- If J_2 were assigned lowest priority, J_1 and J_3 could consume $C_1(2) + C_3(2) = 2 + 4 = 6$ units of processor capacity over $[0, 6)$, thus leaving no execution for J_2 prior to its deadline at time-instant 5.

- If J_3 were assigned lowest priority, J_1 and J_2 could consume $C_1(2) + C_2(2) = 2 + 4 = 6$ units of processor capacity over $[0, 6)$. This leaves 4 units of execution for J_3 prior to its deadline at time-instant 10, which is sufficient for J_3 to execute for $C_3(2) = 4$ time units. *Job J_3 may therefore be assigned the lowest priority.*

Next, the OCBP priority assignment algorithm would consider the instance $\{J_1, J_2\}$, and seek to assign one of these jobs the lower priority:

- If J_1 were assigned lower priority, J_2 could consume $C_2(1) = 2$ units of processor capacity over $[0, 2)$. This leaves 2 units of execution for J_1 prior to its deadline at

time-instant 4, which is sufficient for J_1 to execute for $C_1(1) = 2$ time units. *Job J_1 may therefore be assigned the lowest priority from among $\{J_1, J_2\}$.*

- It may be verified that J_2 *cannot* be assigned the lowest priority from among $\{J_1, J_2\}$. If we were to do so, then J_1 could consume $C_1(2) = 2$ units of processor capacity over $[0, 2)$. This leaves 3 units of execution for J_1 prior to its deadline at time-instant 5, which is not sufficient for J_2 to execute for the $C_2(2) = 4$ time units it needs to complete on time.[2]

The final OCBP priority ordering is therefore as follows. Job J_2 has the greatest priority, job J_1 has the next-highest priority, and J_3 has the lowest priority. It may be verified that scheduling according to these priorities is a correct MC scheduling strategy for the instance $\{J_1, J_2, J_3\}$. □

3.4 Load-Based OCBP Schedulability Test

The following lemma shows that OCBP-schedulability implies MC-schedulability.

Lemma 3.1. *If an instance is OCBP-schedulable on a processor, then it is MC-schedulable on the same processor.*

Proof. Suppose that I is OCBP-schedulable and suppose, after renumbering jobs, that $J_1 \rhd J_2 \rhd \cdots \rhd J_n$. Notice that in every scenario of criticality level χ_k, the criticality level of job J_k, each job J_j has $c_j \leq C_j(\chi_k)$. OCBP-schedulability of I implies that J_k can receive $C_k(\chi_k)$ units of execution before its deadline if each $J_i \in \{J_1, \ldots, J_{k-1}\}$ executes for no more than $C_i(\chi_k)$ units. □

The following theorem provides quantitative load bounds on OCBP scheduling algorithms, and also leads to the speedup factor result of OCBP scheduling algorithm.

[2]We point out that OCBP will *not* actually perform this step of verifying that J_2 cannot be assigned lowest priority since it has already determined, above, that J_1 may be assigned lowest priority.

Theorem 3.2. *If a MC instance I with L criticality levels satisfies*

$$\ell_L + \sum_{k=1}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] \leq 1, \tag{3.1}$$

then I is OCBP-schedulable, thus MC-schedulable.

Proof. This theorem states the fact that the OCBP priority assignment algorithm generates a priority ordering that yields a correct MC scheduling policy for any MC instance I satisfying Condition 3.1. Because Condition 3.1 is easy to implement, it can be used as an efficient schedulability test condition.

We prove this theorem by contradiction, which is that if OCBP priority assignment algorithm does not generate a priority ordering, the instance I must not satisfy Condition 3.1.

Let I denote a *minimal* instance with at most L criticality levels which is MC-schedulable, but the OCBP priority assignment fails to generate a priority ordering.

Without loss of generality, let us assume that $\min_{J_i \in I} A_i = 0$ (i.e., the earliest release time is zero).

Minimality of I implies that there is no time-instant t such that $t \notin \cup_{j=1}^{n} [a_j, d_j]$, otherwise either the jobs with deadline before t or the jobs with release time after t would comprise a smaller instance with the same property. Therefore, there will be no "gap intervals" if we combine all the time windows of the jobs.

Also, minimality implies that there is no job can be assigned the lowest priority by OCBP algorithm. Otherwise, we can recursively remove the lowest-priority job until the OCBP fails to assign the lowest priority. If there are still jobs remaining, these jobs will form a new instance that is not OCBP-schedulable and contradict the fact that I is minimal; if there are no jobs remaining, it will contradict the fact that I is not OCBP-schedulable.

The following lemma states that for all instances that (1) has L criticality levels and (2) has the latest-deadline jobs of criticality $h \neq L$ call all be reduced to an instance with only h criticality levels. Therefore, we only have to prove the case when all the latest-deadline jobs in I are of criticality L.

Lemma 3.2. *Any job in I with the latest deadline must be of criticality L.*

Proof. First, we assume that Theorem 3.2 holds for all instances with criticality level no higher than L. Because when $L = 1$, the Condition 3.1 becomes $\ell_1 \leq 1$, which is trivially true, we can build the proof by induction based on this assumption.

Now suppose that a job J_i with $\chi_i = h < L$ has the latest deadline. Create from I an instance I_h with h levels by "truncating" all jobs with criticality level greater than h to their worst-case level-h scenarios:

$$\begin{aligned} J_j = (a_j, d_j, \chi_j, (C_j(1), \ldots, C_j(L))) &\in I \rightarrow \\ J'_j &= (a_j, d_j, \min(\chi_j, h), (C_j(1), \ldots, C_j(h))) \in I_h. \end{aligned}$$

Clearly, because the WCETs and criticality levels of all jobs in I_h are no greater than those in I. Therefore I_h is a restricted instance of I, and is MC-schedulable as well.

However, it is easy to show that I_h is not OCBP-schedulable, either. The reason is that if J_i can not be assigned the lowest priority in I, J_i and all jobs with criticality level greater than h in I can not be assigned the lowest priority in I_h because they are all in criticality level h in I_h and will all miss their deadlines if assigned the lowest priority (recall that J_i has the latest deadline).

Therefore, by the inductive assumption, because I_h is not OCBP-schedulable, we know that

$$\ell_h + \sum_{k=1}^{h-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{h-1} (\ell_j + 1) \right] \leq 1 \tag{3.2}$$

Then, in order to show that Condition 3.1 does not hold for I either, we can derive that:

$$
\begin{aligned}
&\ell_L + \sum_{k=1}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] \\
=&\sum_{k=h}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] + \ell_L + \sum_{k=1}^{h-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] \\
=&\sum_{k=h}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] + \ell_L + \sum_{k=1}^{h-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{h-1} (\ell_j + 1) \cdot \prod_{j=h}^{L-1} (\ell_j + 1) \right] \\
\geq&\sum_{k=h}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] + \ell_L + \sum_{k=1}^{h-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{h-1} (\ell_j + 1) \right] \\
\geq&\sum_{k=h}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] + \ell_h + \sum_{k=1}^{h-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{h-1} (\ell_j + 1) \right] \\
&\text{(since } \ell_L \geq \ell_h\text{)} \\
>&\sum_{k=h}^{L-1} \left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1} (\ell_j + 1) \right] + 1 \\
&\text{(by assumption 3.2)} \\
>&1
\end{aligned}
$$

This is to say, if I is not OCBP-schedulable and a job J_i with $\chi_i = h < L$ has latest deadline, then I does not satisfy Condition 3.1. □

Lemma 3.2 implies that the theorem holds for those instances I in which the latest-deadline jobs are not of criticality L. In the remainder of this proof we will consider the remaining case, when all the latest-deadline jobs in I are of criticality L.

For each $k \in \{1, \ldots, L\}$, let $d(k)$ denote the latest deadline of any criticality-k job in I: $d(k) = \max_{J_j | \chi_j = k} d_j$. A *work-conserving* schedule on a processor is a schedule that never leaves the processor idle if there is a job available. Consider any such work-conserving schedule on a unit-speed processor of all jobs in I of the scenario in which $c_j = C_j(k)$ for all j. We define Λ_k as the set of time intervals on which the processor is idle before $d(k)$, and λ_k as the total length of this set of intervals.

Claim 3.1. For each k and each $J_j \in I$ with $\chi_j \leq k$ we have $[a_j, d_j] \cap \Lambda_k = \emptyset$.

Proof. Any job J_j with $\chi_j \leq k$ with $[a_j, d_j] \cap \Lambda_k \neq \emptyset$ would meet its deadline if it were assigned lowest priority. Since I is assumed to be non-OCBP schedulable, this implies that $(I \setminus \{J_i\})$ is non-OCBP schedulable on a speed-s processor, contradicting the minimality of I. This completes the proof of the claim. □

It follows that $\Lambda_L = \emptyset$ and $\lambda_L = 0$.

For each $h = 1, \ldots, L$ and $k = 1, \ldots, L$, let

$$c_h(k) = \sum_{J_j | \chi_j = h} C_j(k). \tag{3.3}$$

Notice that by assumption

$$\forall\ k\ \forall h \leq k : c_h(k) = c_h(h). \tag{3.4}$$

From the definition of loads in different criticality levels (Definition 1.9), clearly we have

$$\forall\ k : c_k(k) \leq \ell_k \left[d(k) - \lambda_k \right]. \tag{3.5}$$

Also, any criticality-k scenario, in which each job J_j with criticality $\geq k$ receives exactly $C_j(k)$ units of execution, follows the definition of load in criticality level k:

$$\forall k : \sum_{i=k}^{L} c_i(k) \leq \ell_k \left[d(L) - \lambda_k \right]. \tag{3.6}$$

Instance I is not OCBP-schedulable, which is translated in terms of the introduced notation as:

$$\forall k : \sum_{i=1}^{L} c_i(k) > d(k) - \lambda_k. \tag{3.7}$$

(This follows from Claim 3.1, which shows that no job can execute during the idle intervals Λ_k. Consequently, all the execution on the jobs must have occurred during the remaining $(d(k) - \lambda_k)$ time units.)

Hence, for each k,

$$\begin{aligned} d(k) - \lambda_k &< \sum_{i=1}^{k-1} c_i(k) + \sum_{i=k}^{L} c_i(k) \\ &= \sum_{i=1}^{k-1} c_i(i) + \sum_{i=k}^{L} c_i(k) \quad \text{(by (3.4))} \\ &\leq \sum_{i=1}^{k-1} \ell_i \left[d(i) - \lambda_i \right] + \ell_k \left[d(L) - \lambda_k \right] \quad \text{(by (3.5) and (3.6))} \\ &\leq \sum_{i=1}^{k-1} \ell_i \left[d(i) - \lambda_i \right] + \ell_k d(L). \end{aligned} \tag{3.8}$$

Therefore, for all $k = 1, \ldots, L$,

$$d(k) - \lambda_k < \ell_k d(L) + \sum_{i=1}^{k-1} \ell_i \left[d(i) - \lambda_i \right]. \tag{3.9}$$

Using notation $\delta_k = d(k) - \lambda_k$ (hence $\delta_L = d(L)$ since $\lambda_L = 0$) this yields

$$\delta_k < \ell_k \delta_L + \sum_{i=1}^{k-1} \ell_i \delta_i \tag{3.10}$$

According to Condition 3.10, the summation part in (3.10) can be expanded recursively:

$$\begin{aligned} &\sum_{i=1}^{k} \ell_i \delta_i \\ =&\ell_k \delta_k + \sum_{i=1}^{k-1} \ell_i \delta_i \\ <&\ell_k \left(\ell_k \delta_L + \sum_{i=1}^{k-1} \ell_i \delta_i \right) + \sum_{i=1}^{k-1} \ell_i \delta_i \\ &\text{(by (3.10))} \\ =&\ell_k^2 \delta_L + (\ell_k + 1) \sum_{i=1}^{k-1} \ell_i \delta_i \end{aligned} \tag{3.11}$$

By Condition 3.11, we can represent δ_L by only the loads iteratively:

$$
\begin{aligned}
\delta_L <& \ell_L\delta_L + \sum_{i=1}^{L-1} \ell_i\delta_i \\
<& \ell_L\delta_L + \ell_{L-1}^2\delta_L + (\ell_{L-1}+1)\sum_{i=1}^{L-2} \ell_i\delta_i \\
& \text{(by (3.11))} \\
<& \ell_L\delta_L + \ell_{L-1}^2\delta_L + (\ell_{L-1}+1)\ell_{L-2}^2\delta_L + (\ell_{L-1}+1)(\ell_{L-2}+1)\sum_{i=1}^{L-3} \ell_i\delta_i \\
& \text{(by (3.11) again)} \\
& \dots \\
<& \ell_L\delta_L + \sum_{k=h}^{L-1}\left[\ell_k^2\delta_L \cdot \prod_{j=k+1}^{L-1}(\ell_j+1)\right] + \prod_{j=h}^{L-1}(\ell_j+1)\cdot\sum_{i=1}^{h-1}\ell_i\delta_i \qquad (3.12) \\
& \dots \\
<& \ell_L\delta_L + \sum_{k=2}^{L-1}\left[\ell_k^2\delta_L \cdot \prod_{j=k+1}^{L-1}(\ell_j+1)\right] + \prod_{j=2}^{L-1}(\ell_j+1)\cdot\ell_1\delta_1 \\
<& \ell_L\delta_L + \sum_{k=2}^{L-1}\left[\ell_k^2\delta_L \cdot \prod_{j=k+1}^{L-1}(\ell_j+1)\right] + \prod_{j=2}^{L-1}(\ell_j+1)\cdot\ell_1^2\delta_L \\
& \text{(by (3.8), } \delta_1 < \ell_1\delta_L \text{)} \\
=& \ell_L\delta_L + \sum_{k=1}^{L-1}\left[\ell_k^2\delta_L \cdot \prod_{j=k+1}^{L-1}(\ell_j+1)\right]
\end{aligned}
$$

If we divide both sides of Condition 3.12 by δ_L, we will get exactly

$$
\ell_L + \sum_{k=1}^{L-1}\left[\ell_k^2 \cdot \prod_{j=k+1}^{L-1}(\ell_j+1)\right] > 1. \qquad (3.13)
$$

This is to say, if I is not OCBP-schedulable and all the latest-deadline jobs in I are of criticality L, the reverse form of Condition 3.1 holds, which concludes the proof that Condition 3.1 is sufficient to OCBP-schedulability. □

Corollary 3.1. *If a MC instance I with 2 criticality levels satisfies*

$$\ell_1^2 + \ell_2 \leq 1, \tag{3.14}$$

then I is OCBP-schedulable, thus MC-schedulable.

Proof. Let $L = 2$ in Theorem 3.2, we have

$$\ell_2 + \ell_1^2 \leq 1 \tag{3.15}$$

Therefore I is OCBP-schedulable. □

3.5 Speedup Factors of OCBP Algorithm

The following theorem shows that the OCBP-test is more powerful than the WCR-test according to the speedup criterion.

Theorem 3.3. *If instance I with L criticality levels is MC-schedulable on a given processor, then I is OCBP-schedulable on a processor that is at least s_L times as fast, with s_L equal to the root of the equation $x^L = (1 + x)^{L-1}$.*

Proof. If instance I is MC-schedulable on a speed-1 processor, it must hold that

$$\forall k : \ell_k \leq 1. \tag{3.16}$$

Therefore, on a processor that is s times as fast, if we denote the new loads of I on this speed-s processor as ℓ'_k, we have

$$\forall k : \ell'_k \leq 1/s. \tag{3.17}$$

We notice that the l.h.s. of Condition 3.1 has no negative terms. Therefore, the l.h.s. increases when any ℓ_k increases. Therefore on the speed-s processor,

$$\begin{aligned}
&\ell'_L + \sum_{k=1}^{L-1} \left[\ell'^2_k \cdot \prod_{j=k+1}^{L-1} (\ell'_j + 1) \right] \\
\leq &\frac{1}{s} + \sum_{k=1}^{L-1} \left[\frac{1}{s^2} \cdot \left(\frac{1}{s} + 1 \right)^{L-k-1} \right] \text{ (by (3.17))} \\
= &\frac{1}{s} + \frac{1}{s^2} \cdot \sum_{i=0}^{L-2} \left(\frac{1}{s} + 1 \right)^i \\
= &\frac{1}{s} + \frac{1}{s^2} \left[\frac{s^2 \left(\frac{1}{s} + 1 \right)^L}{s+1} - s \right] \\
= &\frac{1}{s} + \frac{\left(\frac{1}{s} + 1 \right)^L}{s+1} - \frac{1}{s} \\
= &\frac{\left(\frac{1}{s} + 1 \right)^L}{s+1} \\
= &\frac{(1+s)^{L-1}}{s^L}.
\end{aligned} \tag{3.18}$$

If $s \geq s_L$, we will have $(1+s)^{L-1} < s^L$. Thus the l.h.s. of Condition 3.1 will be no more than 1. By Theorem 3.2, the instance I is schedulable on the speed-s processor. □

Theorem 3.4. *The speedup factor of s_L is tight.*

Proof. We now show that the factor s_L is tight by giving instances with L criticality levels that are MC-schedulable on a unit-speed processor, but not OCBP-schedulable on a processor that is less than s_L times as fast.

Consider the following instance consisting of $2L-1$ jobs:

- $J_1 = (0, d_1 = \sigma_1 = 1, 1, (\overbrace{1, 1, \ldots, 1}^{L \text{ times}}))$.
- For each i, $2 \leq i \leq L$, there are two jobs:
  - $J_{2(i-1)} = (0, \sigma_{i-1}, i, (\overbrace{0, 0, \ldots, 0}^{(i-1) \text{ times}}, \underbrace{\sigma_{i-1}, \ldots, \sigma_{i-1}}_{L-(i-1) \text{ times}}))$
  - $J_{2i-1} = (0, \sigma_i, i, (\underbrace{\sigma_i - \sigma_{i-1}, \ldots, \sigma_i - \sigma_{i-1}}_{L \text{ times}}))$, where $\sigma_i > \sigma_{i-1}$.

This instance is MC-schedulable by the following policy. Assign greatest priority to the jobs J_{2i} in reverse order of their indices: $J_{2L}, J_{2(L-1)}, \ldots, J_2$. Consider the scenario in which $c_{2h} > 0$, $h \geq 1$, and $c_{2j} = 0$, $j > h$. Then we execute $J_{2h}, J_{2h+1}, J_{2h+3}, \ldots, J_{2L+1}$ in this order; it is evident that each of them completes by its deadline.

For job J_{2h-1}, $h = 1, \ldots, L$ to be assigned lowest priority in an OCBP-schedule, we would need a speedup factor s of the processor such that

$$\begin{aligned} \frac{(\sigma_L - \sigma_{L-1}) + (\sigma_{L-1} - \sigma_{L-2}) + \cdots + (\sigma_2 - \sigma_1) + \sigma_1 + (1 + \sigma_2 + \cdots + \sigma_{j-1})}{s} &= \\ \frac{\sigma_L + (1 + \sigma_2 + \cdots + \sigma_{j-1})}{s} &\leq \sigma_h. \end{aligned}$$

Hence, for all $h = 1, \ldots, L$, it requires

$$s \geq \frac{\sigma_L + (1 + \sigma_2 + \cdots + \sigma_{j-1})}{\sigma_h}.$$

Thus we have

$$s < \min_{h=1,\ldots,L} \frac{\sigma_L + \sum_{i=1}^{h-1} \sigma_i}{\sigma_h} \tag{3.19}$$

The minimum of the right hand side is maximized if all L terms are equal. Let x be this maximum value. Then for all $k = 1, \ldots, L$,

$$x = \frac{\sigma_L + \sigma_1 + \sigma_2 + \cdots + \sigma_{k-1}}{\sigma_k} = \frac{x\sigma_{k-1} + \sigma_{k-1}}{\sigma_k} = \left(\frac{1+x}{\sigma_k}\right) \sigma_{k-1}.$$

Hence,

$$\sigma_k = \Big(\frac{1+x}{x}\Big)\sigma_{k-1} \ \forall k = 1, \ldots, L \quad \text{which implies} \quad \sigma_L = \Big(\frac{1+x}{x}\Big)^{L-1} \sigma_1 \,.$$

Since, in particular, $x = \frac{\sigma_L}{\sigma_1}$, we have

$$x = \Big(\frac{1+x}{x}\Big)^{L-1}.$$

Therefore, $s \geq s_L$ is necessary for OCBP algorithm to schedule this instance. □

Theorem 3.5. $\mathbf{s_L = \Theta(L/\ln L)}$.

Proof. Rewrite the equation as $x = (1 + 1/x)^{L-1}$ and let x^* be its largest real root. The left hand side (resp., r.h.s.) is increasing (resp., decreasing) in x. The l.h.s. is larger (resp., smaller) than the r.h.s. precisely when $x > x^*$ (resp., $x < x^*$). So if substituting (say) $f(L)$ in place of x gives a l.h.s. larger (resp., smaller) than the r.h.s., it means that $f(L)$ is an upper (resp., lower) bound on x^*.

Substituting $2(L-1)/\ln L$ in place of x, we get for the r.h.s.:

$$(1 + 1/x)^{L-1} \leq e^{(L-1)/x} = e^{(L-1)(\ln L)/2(L-1)} = L^{1/2}$$

(where we have used $1 + y \leq e^y$). The l.h.s. becomes instead $2(L-1)/\ln L$, which is larger than the r.h.s. for all $L \geq 2$. So $x^* \leq 2(L-1)/\ln L$ for all $L \geq 2$.

Substituting $(L-1)/(2\ln L)$ in place of x, we get for the r.h.s.:

$$(1 + 1/x)^{L-1} \geq e^{(L-1)\frac{1}{2x}} = L$$

(where we have used $1 + 2y \geq e^y$ for all $y \in [0, 1.2]$, and assumed $L \geq 3$). The l.h.s. becomes instead $(L-1)/(2\ln L)$, which is smaller than the r.h.s. for all $L \geq 2$. So $x^* \geq (L-1)/(2\ln L)$ for all $L \geq 3$. □

We note that for $L = 2$ in the above theorem, $s_2 = (1 + \sqrt{5})/2$ is equal to the golden ratio $\phi \approx 1.618$; thus the result is a true generalization of an earlier result in (Baruah et al., 2010b). In general, $s_L = \Theta(L/\ln L)$; hence, this priority-based scheduling approach asymptotically improves on the worst-case reservation-based approach (which has a speedup factor $\Theta(L)$) by a factor of $\Theta(\ln L)$ from the perspective of processor speedup factors.

Notice that the proof of the speedup bound for OCBP-schedulability in Theorem 3.3 only uses the clairvoyant-schedulability of the instance, which is a weaker condition than MC-schedulability. It is not possible to get an improved test if the proof of its speedup bound is based on clairvoyant-schedulability alone.

Nevertheless, the question remains if a test other than OCBP can test MC-schedulability within a smaller speedup bound. We do not give a full answer to this question. However, we

can rule out *fixed-priority policies*, that is, policies which execute the jobs in some ordering fixed before execution. This ordering is not adapted during execution, except that we do not execute jobs of criticality level $i < h$ after a scenario was revealed to be a level-h scenario. Such a policy admits a simple representation as a sequence of jobs and we say that an instance I is Π-schedulable if there exists an ordering of jobs Π that is feasible for any non-erroneous scenario.

The following result shows that OCBP is best possible among fixed-priority policies.

Theorem 3.6. *There exist instances with L criticality levels that are clairvoyantly-schedulable, but that are not Π-schedulable for any fixed-priority policy Π on a processor that is less than s_L times as fast, with s_L being the root of the equation $x^L = (1+x)^{L-1}$.*

Proof. Consider an instance with L criticality levels and L jobs:

$$J_1 : (0, 1, 1, (\overbrace{1, 1, \ldots, 1}^{L \text{ times}})),$$

and, for each $i = 2, \ldots, L$,

$$J_i : (0, \sigma_i, i, (\overbrace{\sigma_i - \sigma_{i-1}, \ldots, \sigma_i - \sigma_{i-1}}^{i-1 \text{ times}}, \overbrace{\sigma_i, \ldots, \sigma_i}^{L-i+1 \text{ times}})),$$

where σ_i will be specified later and satisfies $\sigma_{i-1} < \sigma_i$.

For $L = 3$ we have the following example:

$$\begin{array}{lllllll} J_1: & (0, & 1, & 1, & (1, & 1, & 1)) \\ J_2: & (0, & \sigma_2, & 2, & (\sigma_2 - 1, & \sigma_2, & \sigma_2)) \\ J_3: & (0, & \sigma_3, & 3, & (\sigma_3 - \sigma_2, & \sigma_3 - \sigma_2, & \sigma_3)). \end{array}$$

The system is clairvoyantly schedulable as, for each scenario of level i and for each job J_j, $j \geq i$, $\sum_{k=i}^{j} C_k(i) = \sigma_j$. It follows that a schedule that executes job J_i in the interval $[0, \sigma_i]$ and each job J_j, $j > i$, in the interval $[\sigma_{j-1}, \sigma_j]$ is feasible.

We now show that the system is Π-schedulable for a speed-s machine only if $s \geq s_L$ where s_L is the positive real-valued solution of the equation

$$x^L = (x+1)^{L-1}.$$

Each fixed-priority work-conserving policy is a sequence of jobs. Let us consider a sequence where the last scheduled job is J_i and a level i scenario. In this case the overall execution time is $\sum_{j=1}^{L} C_j(i) = \sigma_L + \sum_{j=1}^{i-1} \sigma_j$. Hence the schedule is feasible for a speed-s machine if and only if:

$$s\sigma_i \geq \sigma_L + \sum_{j=1}^{i-1} \sigma_j.$$

By using the same arguments for each possible schedule, it follows that a fixed-priority policy Π system is correct for a speed-s machine if and only if

$$s \geq \min_{1 \leq i \leq L} \left\{ \frac{\sigma_L + \sum_{j=1}^{i-1} \sigma_j}{\sigma_i} \right\}.$$

As we showed in the proof of Theorem 3.4, s_L is the maximum value of s' satisfying the inequality:

$$\min_{1 \leq i \leq L} \left\{ \frac{\sigma_L + \sum_{j=1}^{i-1} \sigma_j}{\sigma_i} \right\} \geq s',$$

hence the system is Π-schedulable for a speed-s machine if and only if $s \geq s_L$. □

The following result shows that OCBP has the best speedup factor among all scheduling policies for instances with 2 criticality levels.

Theorem 3.7. *There exist instances with 2 criticality levels that are clairvoyantly-schedulable, but that are not Π-schedulable for any on-line scheduling policy Π on a processor that is less than $(1+\sqrt{5})/2$ times as fast.*

Proof. Consider an instance with L criticality levels and L jobs ($\phi = (1+\sqrt{5})/2$):

$$\begin{array}{lllll} J_1: & (0, & 1, & 1, & (1, \quad 1)) \\ J_2: & (0, & \phi, & 2, & (\phi - 1, \quad \phi)) \end{array}$$

The system is clairvoyantly schedulable because (1) in each scenario of criticality level 1, a clairvoyantly scheduling policy will execute J_1 first, guarantee J_1 receives 1 time unit in the interval $[0, 1]$, and guarantee J_2 receives $\phi - 1$ time unit in the interval $[1, \phi]$; (2) in each scenario of criticality level 2, a clairvoyantly scheduling policy will only execute J_2 and guarantee J_2 receives ϕ time unit in the interval $[0, \phi]$.

We now show that the system is Π-schedulable for a speed-s machine only if $s \geq \phi$. Because the behavior of the scenario is revealed only at run-time, we assume that in a scenario, J_1 will require at least 1 time unit and J_2 will require at least $\phi - 1$ time units. Before J_2 signals its completion, it is not clear whether the scenario is in criticality level 1 or criticality level 2.

Now we check the time instant when J_2 receives exactly $\phi - 1$ time units by Π on the speed-s machine. If the time instant is earlier than 1, we will let J_2 signal its completion. In this case, the scenario is revealed to be criticality 1 and J_1 must also receive 1 time unit in time interval $[0, 1]$. Therefore, we have $[(\phi - 1) + 1]/s \leq 1$, which leads to $s \geq \phi$. Otherwise if the time instant is later than 1, we will not let J_2 signals its completion and reveal that J_2 will actually require ϕ time units. Because J_2 still needs 1 time unit to complete in the remaining time interval, we know that $1/s \leq \phi - 1$, which also leads to $s \geq \phi$.

Therefore, any on-line scheduling policy Π will require at least a speed-ϕ processor to correctly schedule this instance. This completes the proof. □

3.6 Summary

In this chapter, we focus on the problem of scheduling mixed-criticality jobs. Compared with the mixed-criticality sporadic task sets that we are going to discuss, mixed-criticality job instances consist of finite jobs and certain release-times and deadlines, which make this problem an important base case on which we build the theory of scheduling mixed-criticality sporadic tasks. Also, the problem is interesting in theory because of its intractability in the sense of strong NP-hardness. Therefore, we need to answer two questions: what is the solution to this problem, and how good the solution is.

The first solution that we discuss in this chapter is worst-case reservation scheduling, which is extended from the traditional methods in a straightforward way. The WCR-approach regards all mixed-criticality jobs as traditional real-time jobs. Therefore this approach doesn't have a good performance in the sense of speedup factor because it fails to take advantages of the characteristics of mixed-criticality systems.

The main solution that we discuss is the OCBP algorithm, which provides an applicable solution to the mixed-criticality scheduling problem. We show that OCBP algorithm is sufficient (always providing a correct scheduler) and efficient (always running in polynomial time). The priority-based approach makes this algorithm easy to implement.

A large portion of this chapter focuses on "how approximate OCBP algorithm is". The primary analytic tool we use to answer the question is loads. Theorem 3.2 plays the most important role in the analysis. This theorem is a highly generalized conclusion that can provide quantitative load bounds for a specific number of criticality levels. It also leads to Theorem 3.3 that gives speedup factors for a specific number of criticality levels. It is very important to notice that all these load bounds and speedup factors are aimed at evaluating OCBP algorithm instead of presenting schedulability tests — OCBP algorithm itself is a good schedulability test. Load bounds and speedup factors are used to compare OCBP algorithm and other algorithms (in this chapter it is the WCR-scheduling algorithm) explicitly and quantitatively. Theorem 3.3 and 3.5 show directly that OCBP algorithm is superior to WCR-scheduling algorithm. Theorem 3.6 and 3.7 show that OCBP algorithm is superior than many other potential scheduling algorithms (including *all* other algorithms in two-criticality-level case). Therefore, as a conclusion, OCBP algorithm has very good performance in the sense of approximation.

CHAPTER 4

Scheduling Mixed-Criticality Implicit-Deadline Tasks

In this chapter, we discuss the scheduling of mixed-criticality implicit-deadline tasks with EDF-VD algorithm. The schedulability test and scheduling policy will be described in Section 4.2 and 4.3. The quantitative properties, especially the speedup factor of EDF-VD algorithm, will be discussed in Section 4.4 and 4.5.

4.1 An Overview of Algorithm EDF-VD

In this chapter, we focus on an important special case of sporadic task systems in which each task τ_i satisfies the property that $D_i = T_i$ — such systems, as defined in Subsection 1.3.4, are called *implicit-deadline*, or *Liu & Layland* task systems(Liu and Layland, 1973). Moreover, we focus on the *dual-criticality* case, which means the criticality levels will be no more than 2.[1] The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems was studied in (Baruah et al., 2011a). An algorithm called EDF-VD was proposed that has the same speedup guarantee as above — any task system that can scheduled by an optimal clairvoyant algorithm on a given processor can be scheduled by EDF-VD on a processor that is $(1+\sqrt{5})/2$ times as fast. Moreover, the schedulability test of EDF-VD has polynomial run-time complexity, and the run-time complexity per scheduling decision was logarithmic in the number of tasks. Based on these run-time properties it is evident that EDF-VD, in contrast to the algorithms in (Li and Baruah, 2010a; Guan et al., 2011), can be considered suitable for implementation in actual systems.

[1] As mentioned in Section 1.3.3, we use the notation level LO and HI in this chapter.

The main contribution in this chapter is a more refined analysis of EDF-VD showing that EDF-VD can actually make a better performance guarantee: *any task system that can be scheduled by an optimal clairvoyant algorithm on a given processor can be scheduled by EDF-VD on a processor that is* $4/3$ *times as fast*. This new analysis is based upon some sophisticated new techniques and deep insights that we have recently developed, and represents a substantial improvement over the bound proved in (Baruah et al., 2011a). It was previously shown (Baruah et al., 2012b, Prop. 2) that $(1+\sqrt{5})/2$ is a lower bound on the speedup of any non-clairvoyant algorithm for scheduling collections of independent jobs; it is somewhat surprising that this bound does *not* hold for the more expressive implicit-deadline task model. We also show that no non-clairvoyant algorithm can guarantee to always meet all deadlines on a processor that is less than $4/3$ times as fast as the processor available to the optimal clairvoyant algorithm, thereby proving that EDF-VD is an optimal non-clairvoyant algorithm from the perspective of this metric. In addition, we spell out the details as to how EDF-VD can actually be implemented to have the logarithmic run-time complexity claimed in (Baruah et al., 2011a). We also perform further analysis on the behavior of EDF-VD, deriving a utilization-based schedulability test and exploring its behavior under certain extremal conditions.

Definition 4.1. Let τ denote the MC implicit-deadline sporadic task system that is to be scheduled on a unit-speed preemptive processor. The *EDF-VD algorithm* performs the following actions:

Prior to run-time, EDF-VD performs a schedulability test to determine whether τ can be successfully scheduled by it or not. If τ is deemed schedulable, then an additional parameter, which we call a *modified period* denoted $\hat{T}_i$, is computed for each HI-criticality task $\tau_i \in \tau$. The algorithm for computing these parameters is described in pseudo-code form in Figure 4.1; this pseudo-code is proved correct in Section 4.2.

Observe that it is always the case that $\hat{T}_i \leq T_i$.

Run-time scheduling is done according to the Algorithm EDF, with ***virtual deadlines***: deadlines that EDF-VD computes (in a manner to be described below) and assigns to jobs

Task system $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on a unit-speed preemptive processor.

1. Compute x as follows:
$$x \leftarrow \frac{U_2(1)}{1 - U_1(1)}$$

2. If $\Big(x\,U_1(1) + U_2(2) \leq 1\Big)$ **then**

   $\hat{T}_i \leftarrow x\,T_i$ for each HI-criticality task τ_i

   declare success and **return**

   else declare failure and **return**

Figure 4.1: EDF-VD: The preprocessing phase.

before handing them off to the EDF scheduler. The EDF scheduler will then use these virtual deadlines for the purpose of determining scheduling priority.

These virtual deadlines are assigned as follows. Suppose that a job of task τ_i arrives at time-instant t_a:

- If $\chi_i =$ LO, then this job is assigned a virtual deadline equal to $t_a + T_i$.
- If $\chi_i =$ HI, then this job is assigned a virtual deadline equal to $t_a + \hat{T}_i$.

If some job does execute beyond its LO-criticality WCET without signaling that it has completed execution, the following changes occur:

1. All currently-active LO-criticality jobs are immediately discarded; henceforth, no LO-criticality job will receive any execution.
2. Subsequent run-time scheduling of the HI-criticality tasks (including their jobs that are currently active) continue to be done according to EDF. But the *actual* job deadlines (arrival time plus period) are used.

□

4.2 Schedulability Test: Pre-Runtime Processing

We now provide a detailed description of the pre-runtime processing conducted by EDF-VD. We describe, and prove correct, the strategy used to determine whether a system

is schedulable, and for computing the modified period parameters (the $\hat{T}_k$'s) for systems deemed schedulable. This is also represented in pseudo-code form in Figure 4.1.

As shown in Figure 4.1, EDF-VD first computes a parameter x (the reason why x is assigned this value is derived below — see Expression 4.3) and then assigns values to the $\hat{T}_i$ parameters for all HI-criticality tasks as follows:

$$\hat{T}_i \leftarrow x \times T_i \tag{4.1}$$

Theorem 4.1. *The following condition is sufficient for ensuring that EDF-VD successfully schedules all* LO*-criticality behaviors of* τ*:*

$$x \geq \frac{U_2(1)}{1 - U_1(1)} \tag{4.2}$$

Proof. If EDF is able to schedule all LO-criticality behaviors of the task system obtained from τ by replacing each HI-criticality task τ_i by one with a reduced period, then it follows from the *sustainability* property (Baruah and Burns, 2006) of preemptive uniprocessor EDF that EDF is able to schedule all LO-criticality behaviors of τ as well. Note that scaling down the period of each HI-criticality task by a factor x is equivalent to inflating its utilization by a factor $1/x$. From the utilization-bound result of EDF (Liu and Layland, 1973), we therefore conclude that

$$\begin{aligned} & U_1(1) + \frac{U_2(1)}{x} \leq 1 \\ \Leftrightarrow & \frac{U_2(1)}{x} \leq 1 - U_1(1) \\ \Leftrightarrow & x \geq \frac{U_2(1)}{1 - U_1(1)} \end{aligned}$$

is sufficient for ensuring that EDF-VD successfully schedules all LO-criticality behaviors of τ. □

Algorithm EDF-VD thus chooses for x the smallest value such that Theorem 4.1 is satisfied:

$$x \leftarrow \frac{U_2(1)}{1 - U_1(1)} \tag{4.3}$$

With this value of x, we now determine a sufficient condition for ensuring that EDF-VD successfully meets all HI-criticality deadlines during all HI-criticality behaviors of τ:

Theorem 4.2. *The following condition*[2] *is sufficient for ensuring that EDF-VD successfully schedules all* HI*-criticality behaviors of* τ*:*

$$xU_1(1) + U_2(2) \leq 1 \tag{4.5}$$

Proof. Suppose that τ satisfies Condition 4.2 but EDF-VD cannot meet all deadlines in all HI-criticality behaviors of τ. Let I denote a minimal instance of jobs released by τ, on which a deadline is missed. (By *minimal*, we mean that EDF-VD will meet all deadlines if scheduling any proper subset of I.) Without loss of generality, assume that the earliest job-release in I occurs at time zero, and let t_f denote the instant of the (first) deadline miss — since τ is assumed to satisfy Condition 4.2, this must be the deadline of a HI-criticality job, in a HI-criticality behavior. Let t^* denote the time-instant at which HI-criticality behavior is first flagged (i.e., the first instant at which some job executes for more than its LO-criticality worst-case execution time without signaling that it has completed execution).

We observe that all jobs in I, except perhaps the one that misses a deadline at t_f, experiences some execution; else, the job could be removed from I; this would contradict the assumed minimality of I.

We now introduce some notation for the remainder of this section:

1. For each i, $1 \leq i \leq n$, let η_i denote the amount of execution over the interval $[0, t_f]$ that is needed by jobs in I that are generated by task τ_i.

2. For each i, $1 \leq i \leq n$, let $u_i(\chi)$ denote the quantity $\frac{C_i(\chi)}{T_i}$. (That is, $u_i(\text{LO})$ denotes τ_i's LO-criticality utilization, and $u_i(\text{HI})$ denotes its HI-criticality utilization).

[2]We note here that this theorem is one of the reasons that the results presented in this paper dominate the ones in (Baruah et al., 2011a); the corresponding condition derived in (Baruah et al., 2011a) is

$$x + U_2(2) \leq 1 \tag{4.4}$$

Note that $U_1(1) \leq 1$ is a necessary condition for τ to be schedulable. It is evident that any schedulable system satisfying Condition 4.4 also satisfies Condition 4.5 while the converse is not true: there are task systems satisfying Condition 4.5 that violate Condition 4.4.

3. Let J_1 denote the job with the earliest release time amongst all those that execute in $[t^*, t_f)$. Let a_1 denote its release time, and d_1 its deadline.

Claim 4.1. All jobs that execute in $[t^*, t_f)$ have deadline $\leq t_f$.

Proof. Suppose not. Consider the latest instant t' in $[t^*, t_f)$ when a job with deadline $> t_f$ executes. Only those jobs in I that have release time $\geq t'$ and deadline $\leq t_f$ are sufficient to cause a deadline miss; this contradicts the assumed minimality of I. □

Claim 4.2. Any LO-criticality task τ_i has

$$\eta_i \leq u_i(\text{LO})\big(a_1 + x(t_f - a_1)\big) \tag{4.6}$$

Proof. No LO-criticality job will execute after t^*. For it to execute after a_1, it must have a deadline no larger than J_1's virtual deadline, which is $(a_1 + x(d_1 - a_1))$. Therefore, no LO-criticality job with deadline $> (a_1 + x(t_f - a_1))$ will execute after a_1.

Suppose that some LO-criticality job with deadline $> (a_1 + x(t_f - a_1))$ were to execute, at some time $< a_1$. Let t' denote the latest instant at which any such job executes. This means that at this instant, there were no jobs with effective deadline $\leq (a_1 + x(t_f - a_1))$ awaiting execution. Hence the instance obtained by considering only those jobs in I that have release times $\geq t'$ also misses a deadline; this contradicts the assumed minimality of I. □

Claim 4.3. Any HI-criticality task τ_i has

$$\eta_i \leq \frac{u_i(\text{LO})}{x} a_1 + (t_f - a_1) u_i(\text{HI}) \tag{4.7}$$

Proof. We consider separately the cases when τ_i does not have a job with release time $\geq a_1$, and when it does.

Case: If τ_i does not release a job at or after a_1. We claim that each job of τ_i has a modified deadline $\leq (a_1 + x(t_f - a_1))$. To see why this is so, consider some job with a modified deadline $> (a_1 + x(t_f - a_1))$, and let t' denote the latest instant at which this job executes.

All jobs in I that have release times $\geq t'$ also miss a deadline; this contradicts the assumed minimality of I.

Since each job has a modified deadline $\leq (a_1 + x(t_f - a_1))$, their actual deadlines are all $\leq \frac{a_1}{x} + (t_f - a_1)$. Therefore, their cumulative execution requirement is at most

$$\begin{aligned} & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{LO}) \\ \leq \quad & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \end{aligned}$$

Case: If τ_i releases a job at or after a_1. Let a_i denote the first release $\geq a_1$. The cumulative execution requirement of all jobs of τ_i is at most

$$\begin{aligned} & a_i u_i(\text{LO}) + (t_f - a_i) u_i(\text{HI}) \\ \leq \quad & a_1 u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ & \quad (\text{since } a_1 \leq a_i \text{ and } u_i(\text{LO}) \leq u_i(\text{HI})) \\ \leq \quad & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ & \quad (\text{since } x \leq 1) \end{aligned}$$

Thus in both cases, Condition 4.7 holds. □

Let us sum the cumulative demand of all the tasks over $[0, t_f)$:

$$\begin{aligned} & \sum_{\chi_i = \text{LO}} \eta_i + \sum_{\chi_i = \text{HI}} \eta_i \\ \leq \quad & \sum_{\chi_i = \text{LO}} u_i(\text{LO}) \big(a_1 + x(t_f - a_1)\big) \\ & + \sum_{\chi_i = \text{HI}} \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ = \quad & a_1 \big(U_1(1) + \frac{U_2(1)}{x}\big) \\ & + (t_f - a_1)(x U_1(1) + U_2(2)) \\ \leq \quad & (\text{By choice of } x \text{ [Eqn. 4.2]}, (U_1(1) + \tfrac{U_2(1)}{x}) \leq 1) \\ & a_1 + (t_f - a_1)(x U_1(1) + U_2(2)) \end{aligned}$$

It follows from the infeasibility of this instance that

$$\begin{aligned} & a_1 + (t_f - a_1)(xU_1(1) + U_2(2)) > t_f \\ \Leftrightarrow \quad & (t_f - a_1)(xU_1(1) + U_2(2)) > t_f - a_1 \\ \Leftrightarrow \quad & xU_1(1) + U_2(2) > 1 \end{aligned}$$

Taking the contrapositive, it follows that $(xU_1(1) + U_2(2) \leq 1)$ is sufficient to ensure HI-criticality schedulability by EDF-VD, as is claimed in this theorem. □

We have thus established the correctness of Algorithm EDF-VD: by Theorem 4.1 the value assigned to x ensures the correctness of all LO-criticality behaviors whereas Theorem 4.2 guarantees the correct scheduling of all HI-criticality behaviors.

Observation. Note that Theorem 4.1 requires that $x \geq \frac{U_2(1)}{1-U_1(1)}$, while Theorem 4.2 requires that $x \leq \frac{1-U_2(2)}{U_1(1)}$. When these upper and lower bounds on x are not equal to each other, a pragmatic choice would be to choose a value for x that lies somewhere within the interval (e.g., at the mid-point), rather than at either of the boundaries — this would increase the robustness of the algorithm and its tolerance to, e.g., any arrival jitter.

4.3 Run-time Scheduling Policy

During the execution of the system, jobs are selected for execution according to the following rules:

1. There is a *criticality level indicator* Γ, initialized to LO.
2. While ($\Gamma \equiv$ LO),
   (a) Suppose a job of some task $\tau_i \in \tau$ arrives at time t
      - if $\chi_i \equiv$ LO, the job is assigned a scheduling deadline equal to $t + T_i$.
      - if $\chi_i \equiv$ HI, the job is assigned a scheduling deadline equal to $t + \hat{T}_i$.

   (b) At each instant the waiting job with earliest scheduling deadline is selected for execution (ties broken arbitrarily).

(c) If the currently-executing job executes for more than its LO-criticality WCET without signaling completion, then the behavior of the system is no longer a LO-criticality behavior, and $\Gamma \leftarrow$ HI.

3. Once ($\Gamma \equiv$ HI),

   (a) The scheduling deadline of each HI-criticality job that is currently active is changed to its release time plus the unmodified period parameter (the T_i, not the $\hat{T}_i$) of the task that generated it. That is, if a job of τ_i that was released at some time t is active, its deadline, for scheduling purposes, is henceforth $t + T_i$.

   (b) When a future job of τ_i arrives at some time t, it is assigned a scheduling deadline equal to $t + T_i$.

   (c) LO-criticality jobs will *not* receive any further execution. Therefore at each instant the earliest-deadline waiting job generated by a HI-criticality task is selected for execution(ties broken arbitrarily).

4. An additional rule could specify the circumstances when Γ gets reset to LO. This could happen, for instance, if no HI-criticality jobs are active at some instant in time. (We will not discuss the process of resetting $\Gamma \leftarrow$ LO any further in this document, since this is not relevant to the certification process — LO-criticality certification assumes that the system *never* exhibits any HI-criticality behavior, while HI-criticality certification is not interested in the behavior of the LO-criticality tasks.)

4.3.1 An Efficient Implementation of Run-Rime Dispatching

For traditional (non-MC) sporadic task systems consisting of n tasks, uniprocessor EDF can be implemented efficiently to have a run-time complexity of $O(\log n)$ per event, where an *event* is either the arrival of a job, or the completion of the execution of a job (see, e.g., (Mok, 1988)). A direct application of such implementations can be used to obtain an implementation of the run-time dispatching of EDF-VD that has a run-time of $O(\log n)$ per job-arrival and job-completion event. However, EDF-VD potentially needs to deal with an additional run-time event: the change in the criticality level of the behavior from LO to HI

(this is the event that is triggered at the instant that Γ gets assigned the value HI). Since this event requires that each subsequent scheduling be done according to each HI-criticality task's original deadline, explicitly recomputing priorities according to these original deadlines would take time linear in the number of HI-criticality tasks — in the worst case, $O(n)$ time. We now describe an implementation of EDF-VD's run-time system that has a worst-case run-time of $O(\log n)$ per event for all three kinds of events: job arrival, job completion, and change in the criticality level of the behavior from LO to HI.

Recall that a priority queue supports the operations of inserting (*"insert"*) and deleting the smallest item (*"deleteMin"*) in logarithmic time, and the operation of finding the smallest item (*"min"*) in constant time. In addition, the standard priority queue data structure can be enhanced to support the deletion of a specified item (the *"delete"* operation), also in logarithmic time (see, e.g, (Cormen et al., 2009, Sec. 6.5)). We maintain two such enhance priority queues, Q_{LO} and Q_{HI}. We also use a timer that is used to indicate whether the currently-executing job has executed for more than its LO-criticality WCET (thereby triggering the assignment $\Gamma \leftarrow \text{HI}$).

Initially, $\Gamma \equiv \text{LO}$ and there are three kinds of events to be dealt with: *(1)* the arrival of a job; *(2)* the completion of a job; and *(3)* Γ being assigned the value HI. We consider each separately, below. Suppose that the event occurs at time-instant t_c, and let J_c denote the currently-executing job.

1. A job of task τ_i arrives at time t_c.

   (a) Insert the newly-arrived job into Q_{LO}, prioritized according to its modified scheduling deadline.

   (b) If $\chi_i = \text{HI}$ (i.e., if it is a HI-criticality job), then also insert it into Q_{HI}, prioritized according to its *un*modified (i.e., actual) scheduling deadline.

   (c) If J_c is no longer the minimum job in Q_{LO}, it must be the case that the newly-arrived job has an earlier modified deadline than J_c's modified deadline. In that case, the newly-inserted job becomes J_c, and the timer is set to go off at $t_c + C_i(\text{LO})$ (when this newly-inserted job would exceed its LO-criticality WCET if allowed to execute without interruption).

2. The currently-executing job J_c completes execution at time t_c.

   (a) Delete this job from Q_{LO}, using the *deleteMin* operation supported by priority queue implementations.

   (b) If it was a HI-criticality job, also delete it from Q_{HI} — this would be accomplished by a *delete* operation.

   (c) Set the current-job indicator J_c to denote the new minimum (modified) deadline job — the "minimum" job in Q_{LO}; and set the timer to go off at t_c+ this job's remaining LO-criticality WCET (when the job would exceed its LO-criticality WCET if allowed to execute without interruption).

3. The timer goes off, indicating that the currently-executing job has executed beyond its LO-criticality WCET without signaling completion. The system is therefore now in HI-criticality mode, and we switch to scheduling according to Q_{HI}. Henceforth, all run-time dispatch decisions are taken as indicated by this priority queue.

After Γ becomes HI, no LO-criticality jobs need execute, and HI-criticality jobs are executed according to EDF with their original (unmodified) deadlines. Hence subsequent run-time dispatching is done as for traditional EDF scheduling (as described in, e.g., (Mok, 1988)), with Q_{HI} being the priority queue used for this purpose.

4.4 Some Properties of EDF-VD Algorithm

In this section, we discuss two properties of EDF-VD algorithm. The first property is that EDF-VD algorithm is strictly superior than WCR-scheduling algorithm; the second property is that EDF-VD algorithm successfully schedules a boundary case where $U_2(1) = 0$.

4.4.1 Comparison with Worst-Case Reservation Scheduling

Under the *worst-case reservations* strategy that is widely used in the design of mixed-criticality systems, computing capacity is provisioned to each task at its own criticality level.

That is, the MC task system τ is mapped on to the traditional (non-MC) task system

$$\bigcup_{\tau_i \in \tau} \Big\{ \big(C_i(\chi_i), T_i\big) \Big\}$$

and scheduled using regular EDF. It directly follows from the utilization-bound result of EDF (Liu and Layland, 1973) that the condition

$$U_1(1) + U_2(2) \leq 1 \tag{4.8}$$

is necessary and sufficient for ensuring that EDF can schedule τ to meet all deadlines, provided each LO-criticality job executes for up to its LO-criticality WCET and each HI-criticality job executes for up to its HI-criticality WCET. This covers all LO-criticality and all HI-criticality behaviors of τ.

We now show that Algorithm EDF-VD strictly dominates the worst-case reservations approach: any task system that can be scheduled using worst-case reservations can be scheduled by EDF-VD.

Theorem 4.3. *Any task system τ that is correctly scheduled using worst-case reservations is also correctly scheduled by EDF-VD.*

Proof. Any task system that can be scheduled using worst-case reservations satisfies Condition 4.8 above.

$$\begin{aligned} & U_1(1) + U_2(2) \leq 1 \\ \Rightarrow \quad & \text{(Since } U_2(1) \leq U_2(2)\text{)} \\ & U_1(1) + U_2(1) \leq 1 \\ \Rightarrow \quad & \frac{U_2(1)}{1 - U_1(1)} \leq 1 \end{aligned}$$

From this and Equation 4.3, we conclude that x is assigned a value ≤ 1 by Algorithm EDF-VD.

Now, Theorem 4.2 contains the following sufficient condition for EDF-VD schedulability:

$$xU_1(1) + U_2(2) \leq 1 \ ,$$

which always holds since Condition 4.8 holds and $x \leq 1$.

□

4.4.2 Task Systems τ with $U_2(1) = 0$

It is interesting to analyze the manner in which EDF-VD deals with task systems in which the LO-criticality WCET of each HI-criticality task is equal to zero. (This would be the case for systems in which a "mode change" can be thought to occur when the high-criticality behavior is triggered.)

For such a system τ, observe that $U_2(1) = 0$. Therefore, the value assigned to the scaling parameter x in Step 1 of Figure 4.1 is equal to zero, and the test in Step 2 of Figure 4.1 evaluates to true to all τ with $U_2(2) \leq 1$. Thus EDF-VD can schedule any task system for which $U_2(1) = 0$ that satisfies the condition

$$U_1(1) \leq 1 \text{ and } U_2(2) \leq 1 \ .$$

I.e., EDF-VD can schedule any such system provided the LO-criticality and the HI-criticality behaviors are separately schedulable.

Continuing to analyze the pseudo-code in Figure 4.1 for this special case, we observe that EDF-VD assigns each HI-criticality task τ_i a modified period $\hat{T}_i$ equal to zero. Thus during run-time each job of a HI-criticality task immediately becomes an earliest-deadline (and hence highest-priority) one upon arrival. If it is discovered to have a non-zero execution time, the criticality-level indicator is immediately assigned the value HI (i.e., $\Gamma \leftarrow$ HI), and all LO-criticality jobs are immediately discarded.

4.5 Speedup Factor of EDF-VD Algorithm

The *speedup factor* of an algorithm A for scheduling mixed-criticality systems is defined to be the smallest real number f such that any task system τ that is schedulable on a unit-speed processor by a hypothetical optimal clairvoyant algorithm is successfully scheduled on a speed-f processor by algorithm A. The speedup factor is a convenient metric for comparing the worst-case behavior of different algorithms for solving the same problem: the smaller the speedup factor, the closer the behavior of the algorithm to that of a clairvoyant optimal algorithm.

Theorem 4.4. *The speedup factor of EDF-VD is* $\leq \frac{4}{3}$.

Proof. To prove this theorem, we will show that any MC implicit-deadline sporadic task system that is clairvoyantly schedulable on a speed-$\frac{3}{4}$ processor is schedulable by EDF-VD on a unit-speed processor.

Let b denote an upper bound on both the LO-criticality utilization and the HI-criticality utilization of task system τ:

$$b \geq \max\Big(U_1(1) + U_2(1), U_2(2)\Big) \tag{4.9}$$

By Theorems 4.1 and 4.2, we know that if an x satisfying both theorems exists, there will be no deadline miss. Since Theorem 4.1 requires that

$$\frac{U_2(1)}{1 - U_1(1)} \leq x$$

while Theorem 4.2 requires that

$$x \leq \frac{1 - U_2(2)}{U_1(1)} \ ,$$

we can derive Expression 4.10 below as a sufficient condition for τ to be successfully scheduled using EDF-VD:

$$\begin{aligned}
& \frac{U_2(1)}{1-U_1(1)} \leq \frac{1-U_2(2)}{U_1(1)} \\
\Leftarrow \quad & (\text{Since } U_1(1)+U_2(1) \leq b \Rightarrow U_2(1) \leq b - U_1(1)) \\
& \frac{b-U_1(1)}{1-U_1(1)} \leq \frac{1-U_2(2)}{U_1(1)} \\
\Leftarrow \quad & (\text{Since } U_2(2) \leq b) \\
& \frac{b-U_1(1)}{1-U_1(1)} \leq \frac{1-b}{U_1(1)} \\
\Leftrightarrow & \\
& \big(U_1(1)\big)^2 - U_1(1) + (1-b) \geq 0 \qquad (4.10)
\end{aligned}$$

Now if we set $b \leftarrow \frac{3}{4}$, Expression 4.10 becomes

$$\begin{aligned}
& \big(U_1(1)\big)^2 - U_1(1) + \frac{1}{4} \geq 0 \\
\Leftrightarrow & \\
& \big(U_1(1) - \frac{1}{2}\big)^2 \geq 0
\end{aligned}$$

which is true for all values of $U_1(1)$.

We have thus shown that any task system that is clairvoyant schedulable on a speed-$\frac{3}{4}$ processor is scheduled by EDF-VD to meet all deadlines on a unit-speed processor. It therefore follows that any task system that is clairvoyant schedulable on a unit-speed processor is scheduled by EDF-VD to meet all deadlines on a speed-$\frac{4}{3}$ processor, as claimed by this theorem. □

We now show that EDF-VD is optimal with regard to speedup factor:

Theorem 4.5. *No non-clairvoyant algorithm for scheduling dual-criticality implicit-deadline sporadic task systems can have a speedup bound better than* $\frac{4}{3}$.

Proof. Consider the example task system $\tau = \{\tau_1, \tau_2\}$, with the following parameters, where ϵ is an arbitrarily small number > 0:

τ_i	χ_i	C_i(LO)	C_i(HI)	T_i
τ_1	LO	$1+\epsilon$	$1+\epsilon$	2
τ_2	HI	$1+\epsilon$	3	4

This system is schedulable by a clairvoyant scheduler: EDF would meet all deadlines in LO-criticality behaviors (since $U_1(1) + U_2(1) \leq 1$), while only jobs of τ_2 would get to execute in HI-criticality behaviors.

To see that τ cannot be scheduled correctly by an on-line scheduler, suppose both tasks were to generate jobs simultaneously. It need not be revealed prior to one of the jobs receiving $(1+\epsilon)$ units of execution, whether the behavior is a LO-criticality or a HI-criticality one. We consider two cases.

1. τ_1's job receives $(1+\epsilon)$ units of execution before τ_2's job does. In this case, the behavior is revealed to be a HI-criticality one. But now there is not enough time remaining for τ_2's job to complete by its deadline at time-instant 4.

2. τ_2's job receives $(1+\epsilon)$ units of execution before τ_1's job does. In this case, the behavior is revealed to be a LO-criticality one, in that τ_2's job signals that it has completed execution. But there is not enough time remaining for τ_1's job to complete by its deadline at time 2.

We have thus shown that no non-clairvoyant algorithm can correctly schedule τ. The theorem follows, based on the observation that $\max\Big(U_1(1) + U_2(1), U_2(2)\Big)$ exceeds 3/4 by an arbitrarily small amount. □

4.6 Summary

In this chapter, we focus on a special case of mixed-criticality sporadic tasks — the implicit-deadline tasks. Because the conciseness of the implicit-deadline task model makes it very popular in the real-time system research, we spend a whole chapter discussing the scheduling problem under this model.

Different from the OCBP algorithm in Chapter 3, EDF-VD algorithm is presented as a scheduling policy in Section 4.1 (more run-time details are discussed later in Section

4.3). Therefore, we need to construct the schedulability test for this scheduling policy. In Section 4.2, we describe and prove the schedulability condition in Figure 4.1. The proof is done by considering the maximum time demand over a low-criticality or high-criticality busy interval, while the time demand in the interval is shaped by utilizations. A possible future work may exist because we actually simplified the problem by discarding specific task parameters in the system. If we can bound the time demand by a more detailed description with task parameters, we may get a more accurate schedulability condition. The current result consists of only utilizations — it means that all sporadic task systems are equivalent in the schedulability test if they have identical high/low utilizations.

The speedup factor for EDF-VD algorithm on mixed-criticality implicit-deadline sporadic tasks is 4/3, which is better than the previously shown factor 1.618 (Baruah et al., 2011a). Surprisingly, it is also better than OCBP algorithm's 1.618 factor on mixed-criticality jobs (which is shown to be optimal in two-criticality-level case). The reason is that implicit-deadline model imports more constraints to the system, so we can shape the time demand in an interval more accurately. We show that the 4/3 factor is optimal, which means that it is impossible to discover any on-line scheduling algorithm that has a better speedup factor (but it is possible to find a better algorithm that accepts more task sets that EDF-VD). As a further contribution, we have shown how EDF-VD can be implemented to have a run-time complexity per scheduling decision that is logarithmic in the number of tasks, and thus demonstrated the practical applicability of the algorithm.

CHAPTER 5

Scheduling Mixed-Criticality Arbitrary-Deadline Tasks

In this chapter, we discuss the scheduling of mixed-criticality arbitrary-deadline tasks with EDF-VD algorithm. Because the scheduling policy remains the same as in Chapter 4, we focus on the schedulability test in Section 5.2, and on the speedup factor in 5.3.

5.1 Overview

Here we study the case in which deadlines do not equal periods, the so-called sporadic task system with *arbitrary deadlines*. In this chapter we still only consider task systems that consist of two criticality levels.

In the case that deadlines are not equal to periods, we need more than the notion of utilization in order to determine schedulability. In classical real-time scheduling theory, the *load* of an instance denotes the maximum over all time intervals, of the cumulative execution requirement by jobs of the instance over the interval, normalized by the interval length. Informally, the load of an instance represents a lower bound on the speed of any processor upon which it can meet all deadlines.

As introduced in Section 1.3, a sporadic task system τ can generate infinitely many different instances. Each instance can be seen as an independent collection of jobs where the realized arrival time of job J_{ij} of task τ_j is denoted by a_{ij} and its absolute deadline by $d_{ij} = a_{ij} + d_j$. Then for instance I of the mixed-criticality system we can define three notions of load, analogous to the beforementioned concept from real-time scheduling.

The following definition defines the "worst-case load" of a mixed-criticality system, in addition to the previously defined loads in different criticality levels:

Definition 5.1. The load $\ell(I)$, the LO-load $\ell_1(I)$ and the HI-load $\ell_2(I)$ of a mixed-criticality instance I with two criticality levels are defined according to the following three formulas:

$$\ell(I) = \max_{0 \le t_1 < t_2} \frac{\sum_{J_{ij}: t_1 \le a_{ij} \wedge d_{ij} \le t_2} c_j(\chi_j)}{t_2 - t_1},$$

$$\ell_1(I) = \max_{0 \le t_1 < t_2} \frac{\sum_{J_{ij}: t_1 \le a_{ij} \wedge d_{ij} \le t_2} c_j(1)}{t_2 - t_1},$$

$$\ell_2(I) = \max_{0 \le t_1 < t_2} \frac{\sum_{J_{ij}: \chi_j = 2 \wedge t_1 \le a_{ij} \wedge d_{ij} \le t_2} c_j(2)}{t_2 - t_1}.$$

Informally, $\ell(I)$ is the largest load the system can have if no job is omitted at all; $\ell_1(I)$ is the largest load the system can have if we clairvoyantly know that the system runs in LO-criticality level; $\ell_2(I)$ is the largest load the system can have if we clairvoyantly know that the system runs in HI-criticality level.

These definitions extend in the obvious manner to systems of sporadic tasks. The load of sporadic task system $\ell(\tau)$ is defined to be the largest value that $\ell(I)$ can have, for any instance I generated by τ. We define by the *synchronous arrival sequence* of task system τ the collection of job arrivals in which each task generates a job at time-instant zero, and subsequent jobs arrive as soon as possible. It is well-known (Baruah et al., 1990) that the largest demand over any time interval is reached for this synchronous arrival sequence. Also, note that only values of t_1 and t_2 have to be considered where t_1 is equal to some a_{ij} and t_2 is equal to some d_{ij}. For the interval $[0, lcm\{c_1, \ldots, c_n\})$, there are a pseudo-polynomial number of such intervals $[t_1, t_2)$. □

Thus, for any task system τ, all three loads can be computed in pseudo-polynomial time using well-known techniques for determining the loads of "regular" (i.e., non-MC) sporadic task systems. Specifically, $\ell(\tau)$, ℓ_1 and ℓ_2 are respectively the loads of "regular" sporadic task systems $\{(c_j(\chi_j), d_j, c_j) \,|\tau_j \in \tau\}$, $\{(c_j(1), d_j, c_j) \,|\tau_j \in \tau\}$ and $\{(c_j(2), d_j, c_j) \,|\tau_j \in \tau \wedge \chi_j = 2\}$. In the sequel we will abbreviate $\ell(\tau)$, $\ell_1(\tau)$ and $\ell_2(\tau)$ to ℓ_0, ℓ_1 and ℓ_2.

5.2 Schedulability Test

The following observation gives a necessary condition for any scheduling algorithm to schedule an MC task system.

Claim 5.1. If for any MC task system τ, $\ell_1 > 1$ or $\ell_2 > 1$, the system is not schedulable by any scheduling algorithm.

The relationship between the three loads is given in the next observation.

Claim 5.2. For any MC task system τ, $\ell_0 \leq \ell_1 + \ell_2$.

Algorithm EDF-VD: The EDF-VD algorithm proceeds as follows:

Case 1. If $\ell_0 \leq 1$, schedule all the jobs with EDF scheduler according to their original deadlines;

Case 2. If $\ell_0 > 1$, test if $\ell_1 + \ell_2/2 \leq 1$ and $(\ell_1\ell_2 - 2\ell_1)^2 - 4\ell_1(\ell_1 - \ell_2 + \ell_2^2) \geq 0$. If both inequalities hold, set $x = 1 - \ell_2/2$, scale the relative deadlines of every HI-criticality job J_{ij} from task τ_j from d_j to $\hat{d}_j = xd_j$, and accordingly, the absolute deadlines from $d_{ij} = a_{ij} + d_j$ to $\hat{d}_{ij} = a_{ij} + xd_j$. In LO-criticality level, schedule all the jobs with the EDF scheduler according to the original deadlines of LO-criticality jobs and the virtual deadlines of HI-criticality jobs; when the system's behavior alters to HI (a HI-criticality job uses more than its LO-criticality WCET), cancel all LO-criticality jobs, reset all HI-criticality jobs' deadlines to the original ones, and schedule the HI-criticality jobs with EDF according to their original deadlines immediately.

We prove that the EDF-VD algorithm is correct by the following three theorems.

Theorem 5.1. *If an MC task system τ satisfies $\ell_0 \leq 1$, it is schedulable by EDF-VD.*

Proof. Notice that if every job uses its maximum execution time, which means all LO-criticality jobs use $c_j(1)$ and all HI-criticality jobs use $c_j(2)$, the load of the system will be bounded by ℓ_0. Since EDF is the optimal scheduling strategy on a preemptive uniprocessor, Case 1 of EDF-VD algorithm guarantees that the task system is schedulable if $\ell_0 \leq 1$. □

Theorem 5.2. *If an MC task system τ satisfies $\ell_1 \leq 1$, setting the scaling parameter x in the EDF-VD algorithm such that $\ell_1 \leq x \leq 1$, will guarantee that the system meets all deadlines in* LO*-criticality behavior.*

Proof. To prove that there is no deadline miss, we assume the opposite, which is that a job J_0 misses its deadline in LO-criticality behavior. Let I denote a minimal instance of jobs released by τ on which a deadline is missed. The deadline of job J_0 is denoted as d_0. Now, if we inspect the jobs that can block job J_0 from execution (in the sense that they have an earlier (virtual) deadline and are therefore given priority by EDF), it can only be the LO-criticality jobs that have deadlines no greater than d_0, and the HI-criticality jobs that have virtual deadlines no greater than d_0. Notice that all virtual deadlines can be represented as $\hat{d}_{ij} = a_{ij} + xd_j \geq x(a_{ij} + d_j) = xd_{ij}$. If $\hat{d}_{ij} \leq d_0$, the previous inequality leads to the conclusion that all HI-criticality jobs that can block the execution of job J_0 will have actual absolute deadline no later than d_0/x. Therefore, the cumulative execution requirement of the system until time d_0 is at most $\ell_1 \cdot d_0/x$ in LO-criticality behavior. Because J_0 misses its deadline d_0, it must hold that

$$\begin{aligned} & \frac{d_0}{x}\ell_1 > d_0 \\ \Leftrightarrow \qquad & \ell_1 > x. \end{aligned} \tag{5.1}$$

Since we assumed that job J_0 missed its deadline in LO-criticality behavior, the contrapositive of (5.1),

$$x \geq \ell_1, \tag{5.2}$$

must be a sufficient condition for schedulability in LO-criticality level. □

Theorem 5.3. *If an MC task system τ satisfies $(\ell_1\ell_2 - 2\ell_1)^2 - 4\ell_1(\ell_1 - \ell_2 + \ell_2^2) \geq 0$, then by selecting $x = 1 - \ell_2/2$ as the scaling parameter in the EDF-VD algorithm, the system will not miss any deadline in* HI*-criticality behavior.*

Proof. To prove that there is no deadline miss, we argue by contradiction and assume that a job J_2 misses its deadline at time d_2. Let I denote the minimal instance in which a deadline miss occurs. Therefore, job J_2 must be of HI-criticality. Denote by time t^* the time instant that HI-criticality behavior is first flagged.

Claim 5.3. All jobs J_{ij} that are executed in $[t^*, d_2)$, have actual absolute deadline $d_{ij} \leq d_2$, where d_2 is the deadline of the job that misses its deadline.

Proof. Note that only jobs of HI-criticality level will execute after t^*. Further, after t^* the original deadlines of all jobs are restored. Now suppose there is a job that has a deadline larger than d_2, but is executed somewhere in $[t^*, d_2)$ and let $[t', t'')$ be the last interval in which it was executed. That means that in that interval no jobs with deadlines smaller or equal to d_2 were pending. The instance obtained by only considering the jobs with release time at least t'' also misses a deadline and this contradicts the assumed minimality of I. □

It is crucial to explicitly identify which jobs can block the execution of job J_2. We have seen from Fact 5.3 that all jobs that are executed after t^* can block the execution of J_2 (for their HI-criticality execution time). We define the set of jobs which are executed after t^* as S_2. From the definition $J_2 \in S_2$, and furthermore, the jobs in S_2 (except J_2 itself) are the only jobs that can block the execution of J_2 in HI-criticality. Thus in LO-criticality, the jobs that block execution of jobs in S_2 will consequently block the execution of J_2.

Definition 5.2. Let t be the time instant when the first job in S_2 is released. This job is denoted as J_1 and d_1 is its actual absolute deadline. □

By definition, we know that before time t, there are no jobs in S_2 available. Thus J_2 is also released at or after t. And from Fact 5.3, all jobs in S_2 have deadlines no later than d_2, which implies $d_1 \leq d_2$.

In order to calculate the maximum possible execution requirement that blocks execution of J_2, we present the following lemma.

Lemma 5.1. *Any job J_{ij} that is in the minimal instance I and is able to block the execution of J_2 in* LO*-criticality behavior must have absolute deadline $d_{ij} \leq t + x(d_2 - t)$ if it is a* LO*-criticality job, and virtual absolute deadline $\hat{d}_{ij} \leq t + x(d_2 - t)$ if it is a* HI*-criticality job.*

Proof. Note that no LO-criticality job J_{ij} with $d_{ij} > \hat{d}_1$ will execute at or after t. That would mean that somewhere in the interval $[t, t^*)$ there is no job with deadline at most $\hat{d}_1$ awaiting execution. This contradicts the fact that J_1 is in S_2 (and thus receives execution after t^*) and hence is not finished before t^*.

Suppose there is a LO-criticality job J_{ij} with $d_{ij} > \hat{d}_1$ that executes somewhere before t. Denote by t' the latest moment in time where it executes. At this point there are no available jobs with deadline at most $\hat{d}_1$ and the instance obtained by only considering jobs released after t' also misses a deadline. That contradicts the assumed minimality of I.

So, any LO-criticality job J_{ij} must have a deadline $d_{ij} \leq \hat{d}_1 = t + x(d_1 - t) \leq t + x(d_2 - t)$, where the last inequality comes from Fact 5.3.

Suppose that there is a HI-criticality job J_{ij} with virtual deadline $\hat{d}_{ij}$, that is released at time $a_{ij} \leq t$. Since its *actual* deadline is at most d_2, its virtual deadline is at most $a_{ij} + x(d_2 - a_{ij})$ and this is no more than $t + x(d_2 - t)$. Hence, such a job has $\hat{d}_{ij} \leq t + x(d_2 - t)$.

If its release time $a_{ij} > t$, the job can only receive execution in LO-criticality level if its virtual deadline is at most the virtual deadline of J_1. Hence, $\hat{d}_{ij} \leq \hat{d}_1 = t + x(d_1 - t) \leq t + x(d_2 - t)$. □

Now, similar to the concept of S_2, we define the set of LO-criticality jobs that are able to block execution of jobs in S_2 (which includes J_2) as S_1^1, and the set of HI-criticality jobs that are only executed before t^* and that are able to block the execution of jobs in S_2 as S_1^2.

Definition 5.3. Let c_1^1 be the sum of execution requirements of jobs in S_1^1; c_2 the sum of execution requirements of jobs in S_2; and let c_1^2 be sum of execution requirements of jobs in S_1^2. □

Definition 5.4. Let $d = d_2 - t$. This means that d is the length of the time interval where jobs from S_2 exist. □

From Lemma 5.1, we directly get the following two inequalities

$$\begin{cases} c_1^1 & \leq \ell_1(t + xd) \\ c_1^1 + c_1^2 & \leq \ell_1(t/x + d). \end{cases} \tag{5.3}$$

The second inequality holds because $a_{ij} + xd_j = \hat{d}_{ij} \leq t + xd$ gives that $d_j \leq (t + xd)/x$, similar to the conclusion we get in the proof of Theorem 5.2.

By definition of ℓ_2, we also obtain another set of inequalities

$$\begin{cases} c_2 & \leq \ell_2 d \\ c_2 + c_1^2 & \leq \ell_2(t/x + d). \end{cases} \tag{5.4}$$

Since J_2 misses its deadline $d_2 = t + d$ as we assumed, we have $c_1^1 + c_1^2 + c_2 > d_2$, which is equivalent to $c_1^1 + c_1^2 + c_2 - t > d$. This is to say, there exist certain d and t such that for given L and x, $c_1^1 + c_1^2 + c_2 - t > d$ holds. Without loss of generality, we assume that d is constant (since all other values can be scaled proportionally), then the maximum value of $c_1^1 + c_1^2 + c_2 - t$ for all possible t will be greater than d.

In order to get the maximum value of our object function $o(t) = c_1^1 + c_1^2 + c_2 - t$, from inequalities (5.3) and (5.4), we get

$$o(t) \leq \begin{cases} f(t) & = \ell_1(t + xd) + \ell_2(t/x + d) - t \\ g(t) & = \ell_1(t/x + d) + \ell_2 d - t. \end{cases} \tag{5.5}$$

Note that since $\ell_1 + \ell_2 > 1$ (otherwise it must be the case that $\ell_0 \leq \ell_1 + \ell_2 \leq 2 \times 0.5 = 1$, according to Observation 5.2) and $\ell_1/x \leq 1$ (because of Theorem 5.2) We obtain the following two inequalities

$$\begin{cases} \dfrac{\partial f(t)}{\partial t} & = \ell_1 + \ell_2/x - 1 \geq \ell_1 + \ell_2 - 1 > 0 \\ \dfrac{\partial g(t)}{\partial t} & = \ell_1/x - 1 \leq 1 - 1 = 0, \end{cases} \tag{5.6}$$

The inequalities in (5.6) show that when t increases, $f(t)$ increases and $g(t)$ decreases, for $t \in (0, +\infty)$. Thus $\max[o(t)]$ can not exceed the intersection of $f(t)$ and $g(t)$, otherwise at least either the inequalities in (5.3) or in (5.4) will be violated. We denote the value of t where $f(t) = g(t)$ as γ.

It is easy to find that

$$\gamma = \frac{\ell_1(1-x)xd}{\ell_2 - \ell_1(1-x)}, \tag{5.7}$$

and therefore

$$\max\left[o(t)\right] \leq o(\gamma) = \frac{\ell_2^2 + \ell_1\ell_2 x - \ell_1 x + \ell_1 x^2}{\ell_2 - \ell_1 + \ell_1 x} d \tag{5.8}$$

That is to say, if J_2 misses its deadline as we assumed, it must hold that $o(\gamma) > d$ for given ℓ_1, ℓ_2 and x. Or equivalently,

$$\ell_1 x^2 + (\ell_1\ell_2 - 2\ell_1)x + \ell_1 - \ell_2 + \ell_2^2 > 0. \tag{5.9}$$

However, x is a value of our choice between $(0, 1)$. We can select $x = -(\ell_1\ell_2 - 2\ell_1)/2\ell_1 = 1 - \ell_2/2 \in (0, 1)$ to get the minimum value of the left side of inequality (5.9). If inequality $o(\gamma) > d$ must hold, we need to let

$$(\ell_1\ell_2 - 2\ell_1)^2 - 4\ell_1(\ell_1 - \ell_2 + \ell_2^2) < 0, \tag{5.10}$$

so that $o(\gamma) > d$ no matter what value we choose for x.

Finally we reach our conclusion that if J_2 misses its deadline for certain t and d as we assumed, we must have inequality (5.10) to hold to assure that $o(\gamma) \geq \max\left[o(t)\right] > d$. Because we assumed that a job would miss its deadline in HI-criticality behavior, the contrapositive of (5.10),

$$(\ell_1\ell_2 - 2\ell_1)^2 - 4\ell_1(\ell_1 - \ell_2 + \ell_2^2) \geq 0 \tag{5.11}$$

is sufficient to guarantee no deadline miss in HI-criticality level if we select $x = 1 - \ell_2/2$.

□

Theorem 5.4. *If for any MC task system τ, $\ell_1+\ell_2/2 \leq 1$ and $(\ell_1\ell_2-2\ell_1)^2-4\ell_1(\ell_1-\ell_2+\ell_2^2) \geq 0$, the system is schedulable by the EDF-VD algorithm.*

Proof. By Theorem 5.3, we can select $x = 1 - \ell_2/2$ to guarantee no deadline miss in HI-criticality level. By Theorem 5.2, $x = 1 - \ell_2/2 \leq \ell_1$ can guarantee no deadline miss in LO-criticality level. □

Theorem 5.5. *If for any MC task system τ, $\ell_0 \leq 1$ or $\max\{\ell_1, \ell_2\} \leq 4 - 2\sqrt{3}$, the system is schedulable by the EDF-VD algorithm.*

Proof. If $\ell_0 \leq 1$, by Theorem 5.1, the system is schedulable. Otherwise, we can assume that $L > 0.5$(otherwise it must be the case that $\ell_0 \leq L + L \leq 2 \times 0.5 = 1$, according to Observation 5.2). Now if we define the left side of (5.11) as $\mathcal{L}(\ell_1, \ell_2)$, it's easy to verify that

$$\begin{cases} \dfrac{\partial \mathcal{L}(\ell_1,\ell_2)}{\partial \ell_1} & = 2\ell_2[\ell_1(\ell_2-4)-2\ell_2+2] < 0 \\ \dfrac{\partial \mathcal{L}(\ell_1,\ell_2)}{\partial \ell_2} & = 2\ell_1[\ell_1(\ell_2-2)-4\ell_2+2] < 0. \end{cases} \tag{5.12}$$

Therefore, if we define $L = 4 - 2\sqrt{3}$, the minimum value of $\mathcal{L}(\ell_1, \ell_2)$ is reached at (L, L). Hence we have

$$\mathcal{L}(\ell_1, \ell_2) \geq \mathcal{L}(L, L) = (L^2 - 2L)^2 - 4L^3 = 0. \tag{5.13}$$

Obviously $\ell_1 + \ell_2/2 \leq 1$. By Theorem 5.4, the system is schedulable by EDF-VD algorithm.

□

5.3 Speedup Factor Result

Theorem 5.6. *If an MC task system τ is schedulable on a given processor, it is schedulable by the EDF-VD algorithm on a processor that is $1 + \frac{1}{2}\sqrt{3} \approx 1.866$ times faster.*

Proof. If an MC task system τ is schedulable on a given processor, by Observation 5.1, ℓ_1 and ℓ_2 on the given processor are no greater than 1. Thus on the $(1+\frac{1}{2}\sqrt{3})$-speed processor, its $\ell_1'(T)$ and $\ell_2'(T)$ are no greater than $1/(1+\frac{1}{2}\sqrt{3}) = 4-2\sqrt{3}$. By Theorem 5.5, this task system is schedulable on the $(1+\frac{1}{2}\sqrt{3})$-speed processor. □

Theorem 5.7. *There exists a sporadic task system τ with $\ell_1 = \ell_2 = 4-2\sqrt{3}+\epsilon$ (with $\epsilon > 0$ arbitrarily small) which can not be scheduled by the EDF-VD algorithm.*

Proof. Again, we define $L = 4-2\sqrt{3}$, N is a very large positive integer and M is a very large positive integer, even compared to N. We define a task system τ as follows (every task is represented respectively as $([c_j(1), c_j(2)], d_j, c_j, \chi_j)$)

$$
\begin{aligned}
T = \{\tau_1 = \quad & ([L, L], 1-\epsilon, && +\infty, 1),\\
\tau_2 = \quad & ([0, L+\epsilon], 1, && +\infty, 2),\\
\tau_{3_1} = \quad & ([(\sqrt{L}-L)/N, (\sqrt{L}-L)/N], 1+(1/\sqrt{L}-1)/N, && +\infty, 2),\\
\tau_{3_2} = \quad & ([(\sqrt{L}-L)/N, (\sqrt{L}-L)/N], 1+2(1/\sqrt{L}-1)/N, && +\infty, 2),\\
\cdots, & \\
\tau_{3_i} = \quad & ([(\sqrt{L}-L)/N, (\sqrt{L}-L)/N], 1+i(1/\sqrt{L}-1)/N, && +\infty, 2),\\
\cdots, & \\
\tau_{3_N} = \quad & ([(\sqrt{L}-L)/N, (\sqrt{L}-L)/N], 1/\sqrt{L}, && +\infty, 2),\\
\cdots, & \\
\tau_{3_M} = \quad & ([(\sqrt{L}-L)/N, (\sqrt{L}-L)/N], 1+M(1/\sqrt{L}-1)/N, && +\infty, 2)\}.
\end{aligned}
$$

It is easy to verify that ℓ_1 and ℓ_2 of τ are both $L+\epsilon$, by considering the synchronous arrival sequence, i.e., all tasks release a job at time 0 and subsequent job are release as soon as permitted by the problem parameters. If we only consider τ_1 and τ_2, the claim is trivial. If we take τ_{3_1}, τ_{3_2}, $\cdots$, and τ_{3_i} into consideration, we can see the LO-criticality time demand in time interval $[0, 1+i(1/\sqrt{L}-1)/N)$ is

$$\ell_1 \geq \frac{L + i(\sqrt{L} - L)/N}{1 + i(1/\sqrt{L} - 1)/N} = L.$$

A similar inequality holds for ℓ_2. Therefore ℓ_1 and ℓ_2 of τ are both $L + \epsilon$.

Now let us assume that we have selected a scaling factor x. We would like to show that the system is not schedulable with this (arbitrary) given x.

Given x, we assume that τ_2 releases its first job at time $1 - x$, and all other tasks release their first job at time 0 . Then, the actual deadline of the first job of τ_2 (denoted as J_2) is $d_2 = 1 - x + 1 = 2 - x$, and the virtual deadline will be $d_2' = (1 - x) + x \times 1 = 1$. Because the virtual deadline is greater than the deadline of the first job of τ_1 (denoted as J_1), which is $d_1 = 1 - \epsilon$, J_2 will wait until J_1 finishes.

Also, we would like to know how many jobs in the set $S_3 = \{\tau_{3_1}, \cdots, \tau_{3_i}, \cdots, \tau_{3_N}, \cdots, \tau_{3_M}\}$ will execute before d_2. If N is sufficiently large, we can represent the LO-criticality execution requirement of S_3 in time interval $[0, t)$ where $t \geq 1$ as $Lt - L$ because we can always find an i such that $1 + i(1/\sqrt{L} - 1)/N$ is sufficiently close to t, and we only have to subtract L which is the WCET of J_1.

We know that every job in S_3 with $xd_i < d_2 = 1$ (i.e., $d_i < 1/x$) will be executed before J_2. As stated in previous paragraph, the execution requirement for this time interval $[0, 1/x)$ will be $L/x - L$.

Now let us calculate the execution requirement until time d_2. It will be $L + L + \epsilon + L/x - L = L + L/x + \epsilon$. Also, $d_2 = 2 - x$ as we computed. It is obvious that $L + L/x + \epsilon - (2 - x) = L + L/x + x - 2 + \epsilon \geq L + 2\sqrt{L} - 2 = 4 - 2\sqrt{3} + 2(\sqrt{3} - 1) - 2 + \epsilon = \epsilon > 0$. Hence, for this arbitrarily given x, we can always release the jobs carefully so that in the time interval $[0, 2 - x)$, the execution requirement is greater than the length of the interval (intuitively, smaller x will take more jobs in S_3 into account, and larger x will reduce the length of $[0, 2 - x)$ which eventually causes a deadline miss, too). This implies that is is impossible to schedule τ using the EDF-VD algorithm. □

5.4 Summary

We explore the arbitrary-deadline sporadic task, which is a more generalized task model in this chapter. The run-time scheduling policy keeps the same as in Chapter 4 — we still use EDF-VD algorithm which applies virtual deadlines to high-criticality, while a new schedulability test is applied to this new case. We maintain the idea for constructing the schedulability test — considering the maximum time demand over a low-criticality or high-criticality busy interval. The schedulability test becomes more complicated as the shaping of time demands over an interval becomes harder by loads, compared with by utilizations.

The speedup factor for EDF-VD algorithm on mixed-criticality arbitrary-deadline tasks is 1.866. We show that this factor is the best we can get for EDF-VD algorithm. No better factor (with a low bound of 1.618 shown in Section 3.5) can be achieved unless the current version of EDF-VD algorithm is modified or better algorithms are invented.

CHAPTER 6

Other Contributions

In this chapter, we briefly introduce other contributions including the extension of OCBP algorithm on recurrent tasks and the extension of EDF-VD algorithm on multiprocessor platforms.

6.1 OCBP Algorithm on Mixed-Criticality Recurrent Tasks

In Chapter 3, we considered the scheduling of mixed-criticality job instances: workloads that are specified as collections of independent jobs rather than as systems of recurrent tasks. Though OCBP is designed to work on mixed-criticality jobs, it can also be extended to support mixed-criticality sporadic task systems.

The two main obstacles to apply OCBP algorithm to sporadic task systems are: (1) The OCBP algorithm has two phases: an off-line phase during which priorities are computed for all the jobs, followed by the run-time phase which deploys priority-based dispatching using the priorities assigned during the off-line phase. But since any instance generated by a sporadic task system may contain infinitely many jobs, the off-line procedure for computing all the priorities prior to runtime is not guaranteed to terminate. (2) The algorithm for determining OCBP priorities requires the complete specification of all the jobs in the instance. However, under the (reasonable) assumption that our run-time scheduling algorithm is not clairvoyant, we do not know this information beforehand for sporadic task systems: although we may know a lower bound on the release times of jobs, a jobs exact release time only becomes known when it is actually released.

We deal with the first of these problems — potentially infinitely many jobs — by only assigning priorities, at each instant in time, to those jobs that arrive during the current

busy interval, where the busy interval refers to a maximal continuous interval of time during which the processor is not idled. (This is reasonable: since OCBP scheduling never idles the processor while there are jobs awaiting execution, scheduling decisions made within a particular busy interval are not impacted by the priorities assigned to jobs arriving outside that busy interval.) For any sporadic task system with all loads ℓ_k strictly less than one, prior techniques from real-time scheduling theory can be applied to bound the maximum length of the longest busy interval, and thereby determine the largest collection of jobs that can possibly execute before the processor is idled.

Example 6.1. This example briefly shows the procedure to calculate the longest busy interval for a dual-criticality sporadic task system τ. The loads ℓ_1 and ℓ_2 are the same as defined in Definition 5.1. The technique can be plainly generalized to multiple criticality level cases. We note that this is a pessimistic bound that is to show that such bounds exist, rather than to compute the tightest bound. Less rough estimation can be obtained by more advanced techniques (such as in (George et al., 1996; Ripoll et al., 1996)).

In OCBP algorithm, no LO-criticality job is executed once any job executes for more than its LO-criticality WCET. Let us therefore consider the longest busy interval as being comprised of two parts: (i) from the beginning of the busy interval up to the instant (if any) at which some job executes for more than its LO-criticality WCET, and (ii) from that instant to the end of the busy interval. Without loss of generality, we assume that the busy interval starts at time-instant zero, some job executes for more than its LO-criticality WCET at time-instant x_1, and the busy interval ends at time-instant $x_1 + x_2$.

Let $D_{\max}$ denote the largest deadline of any task in τ: $D_{\max} = \max_{\tau_i \in \tau}\{D_i\}$. All jobs executed over $[0, x_1)$ have their release times and deadlines within the interval $[0, x_1 + D_{\max})$; hence

$$x_1 \leq \ell_1(D_{\max} + x_1) \tag{6.1}$$

$$\Leftrightarrow \quad x_1 \leq \frac{\ell_1}{1-\ell_1} \times D_{\max} \tag{6.2}$$

Since all jobs executing during $[x_1, x_1 + x_2)$ have their release times and deadlines within the interval $[0, x_1 + x_2 + D_{\max}$, it must be the case that

$$\begin{aligned} & x_2 \leq \ell_2(x_1 + x_2 + D_{\max}) \\ \Leftrightarrow & x_2 \leq \ell_2 \left(\frac{\ell_1}{1 - \ell_1} \times D_{\max} + x_2 + D_{\max} \right) \text{ (by (6.1))} \\ \Leftrightarrow & x_2 \leq \frac{\ell_2}{(1 - \ell_1)(1 - \ell_2)} \times D_{\max} \end{aligned} \tag{6.3}$$

The length of the longest busy interval is then bounded from above by $x_1 + x_2$. Under the assumption that ℓ_1 and ℓ_2 are both bounded from above by a constant strictly less than one, this is easily seen to be pseudopolynomial in the representation of τ. Once the length of the longest busy period has been bounded as above, it is straightforward to bound which jobs could have arrived within this interval — there will be at most pseudopolynomially many such jobs. □

To deal with the second problem — job release times (and hence deadlines) not known in advance, we assign priorities under the assumption that all jobs in the current busy interval are released as soon as legally permitted to do so under the constraints of the sporadic task model. Actually, for the purposes of assigning the priorities, we can assume that all these jobs are "early-released", which means that they are immediately available in the beginning of the busy interval. We don't have to simulate the behavior of the schedule as we propose in Section 3.3. If we represent a job collection in the longest busy interval of a mixed-criticality system τ as I and assume that the busy interval starts at time t_0, a job $J_k \in I_\tau$ can be assigned the lowest priority if

$$\sum_{\forall i: J_i \in I} C_i(\chi_k) \leq (d_k - t_0) \tag{6.4}$$

Identical to the method we described in Section 3.3, we include all jobs in the longest busy interval in I in the beginning, and repeatedly apply the algorithm to the new collection of jobs excluding the lowest priority job, until a full priority list is generated or the iteration is terminated.

During runtime, we will monitor the actual job-release times; as long as they conform to the ones we had used in assigning priorities, we need do nothing. When they do not so conform, we will, under some circumstances, need to re-compute the priorities assigned to some of the jobs. The only circumstances that need priority re-computation are: the processor is idle, or the execution of some lesser-priority job J_x is preempted due to the release of some greater-priority job J_y. We use the following dual-criticality example to illustrate our method, which can be generalized to multiple criticality cases.

Example 6.2. A detailed dispatching procedure for a dual-criticality sporadic task system can be described as follows: at each instant, the job J_i with the lowest priority that has been released but not yet signalled completion is selected for execution. This continues until one of the following events has occurred:

1. Some job J_i executes for more than $ci(\text{LO})$ without signalling that it has completed execution. This implies that the system is now in HI-criticality mode, and HI-criticality jobs are no longer required to complete by their deadlines. We may therefore discard all LO-criticality jobs. It follows from the correctness of the OCBP priority assignment at to that all HI-criticality jobs that will arrive during the current busy interval are guaranteed to complete by their deadlines.

2. Under our priority-based scheduling model, the processor is idled at some time-instant t only if all jobs that had arrived prior to t have completed execution by time-instant t. If this happens, the current busy interval has ended, and priorities that were assigned at t_0 to jobs that ended up not arriving during this busy interval are "canceled". We await the release of some job, which will signal the start of a new busy interval — at that time, we will recompute the priorities of all jobs that could possibly be scheduled during that busy interval.

3. The execution of some lesser-priority job J_x is preempted due to the release of some greater-priority job J_y, say at time-instant t_1. We must recompute the priority list at this point in time. It is proved that this recomputation can always generate a new priority list (or equivalently, the schedule according to OCBP so far doesn't make the

system unschedulable even though some tasks arrive late). We will not discuss the detailed proof here. □

The main theorem in this section is stated below:

Theorem 6.1. *Given an MC sporadic task system τ, if the instance I_τ which consists of the jobs in the longest busy interval of τ is OCBP-schedulable on a processor, then τ is MC-schedulable (and thus schedulable) on the same processor.*

This theorem shows that there is an algorithm that successfully schedules any mixed-criticality sporadic task system whose longest busy interval is OCBP schedulable, or satisfies the load condition described in Theorem 3.2. The algorithm has a pseudo-polynomial time complexity in both pre-process and run-time stage, which is relatively computational-expensive compared with EDF-VD algorithm in Chapter 5. However, the algorithm has maintained the optimal speedup factor 1.618 in dual-criticality case, which is superior than EDF-VD.

6.2 Multiprocessor Mixed-Criticality Scheduling

With the modernistic trend of implanting real-time systems on multiprocessor platforms, it is of significance to discover the possibility of scheduling mixed-criticality systems on multiprocessors. The EDF-VD algorithm is extended to both *global* multiprocessor scheduling (tasks and jobs can migrate among processors) and *partitioned* multiprocessor scheduling (tasks are assigned to dedicated processors).

6.2.1 Global Mixed-Criticality Scheduling

Our global mixed-criticality scheduling approach extends the EDF-VD uniprocessor mixed-criticality scheduling algorithm to multiprocessors, by applying a previously-proposed multiprocessor global scheduling algorithm called fpEDF (Baruah, 2004), that was designed for non mixed-criticality systems.

Algorithm fpEDF (Baruah, 2004) is a global EDF-based algorithm for scheduling systems of non mixed-criticality implicit-deadline sporadic tasks upon identical multiprocessor

platforms. Suppose that "regular" (i.e., non-MC) implicit-deadline sporadic task system τ is to be scheduled on m unit-speed processors. During run-time all jobs of tasks in τ that have utilization greater than $1/2$ are assigned highest priority, and the remaining tasks' jobs are assigned priorities according to their deadlines (as in "regular" EDF).

The following theorem is a corollary from Theorem 4 in (Baruah, 2004):

Theorem 6.2. *If a task system cannot be scheduled by Algorithm fpEDF on m unit-speed processors, then it cannot be scheduled by preemptive uniprocessor EDF on a processor of speed $(m+1)/2$.*

Now we provide a high-level overview of the global mixed-criticality scheduling algorithm. Let $\tau = \{\tau_1, \ldots, \tau_n\}$ denote the MC implicit-deadline sporadic task system that is to be scheduled on m unit-speed preemptive processors. Our approach to scheduling τ can be thought of as a three-phased one.

During the *pre-processing phase*, a schedulability test is performed to determine whether τ can be successfully scheduled by our algorithm or not. If τ is deemed schedulable, then an additional parameter, which we call a *modified period* denoted $\hat{T}_i$, is computed for each HI-criticality task $\tau_i \in \tau$. We will see the details for the pre-processing phase later.that $\hat{T}_i \leq T_i$.

Initial run-time scheduling is done according to previously described Algorithm fpEDF (Baruah, 2004). Since fpEDF is defined for regular, rather than mixed-criticality, task systems, we must map the mixed-criticality tasks in τ to regular tasks. This is done as follows: each LO-criticality task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ is mapped to a regular implicit-deadline task $(C_k(\text{LO}), T_k)$, while each HI-criticality task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ is mapped to a regular implicit-deadline task $(C_k(\text{LO}), \hat{T}_k)$, where the $\hat{T}_k$'s are the modified periods computed during the pre-processing phase. It follows from the *sustainability* property (Baruah and Burns, 2006; Baker and Baruah, 2008) of Algorithm fpEDF that if Algorithm fpEDF is able to schedule this regular implicit-deadline sporadic task system then it is able to schedule all LO-criticality behaviors of the MC implicit-deadline task system τ.

If some job does execute beyond its LO-criticality WCET without signaling that it has completed execution, we enter the *third phase* of the algorithm, and the following changes occur.

1. All currently-active LO-criticality jobs are immediately discarded; henceforth, no LO-criticality job will receive any execution.

2. Subsequent run-time scheduling of the HI-criticality tasks (including their jobs that are currently active) are done according to Algorithm fpEDF. In order to do so, we must once again map these HI-criticality tasks to regular implicit-deadline tasks. This is done as follows: each HI-criticality MC task $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$ in τ is mapped to a regular implicit-deadline task $(C_k(\text{HI}), T_k - \hat{T}_k)$.

We now specify what happens during the **pre-processing phase**. The idea behind our schedulability test (pre-processing phase) is to ensure that there is sufficient computing capacity available between this time-instant t^* and the deadline of each currently-active HI-criticality job, to be able to execute all these jobs for up to their HI-criticality WCET's by their respective deadlines. This is ensured by the manner in which the modified periods (the $\hat{T}_k$ parameters) are computed. We will compute modified period values to ensure that the following two properties are satisfied:

1. All jobs of all tasks will meet their modified deadlines in any LO-criticality behavior of the system (i.e., if no job executes beyond its LO-criticality WCET). That is, the collection of "regular" (non-MC) tasks

$$\Big(\bigcup_{\chi_i=\text{LO}} \Big\{ (C_i(\text{LO}), T_i) \Big\} \Big) \bigcup \Big(\bigcup_{\chi_i=\text{HI}} \Big\{ (C_i(\text{LO}), \hat{T}_i) \Big\} \Big) \tag{6.5}$$

   is scheduled by Algorithm fpEDF to always meet all deadlines on the available m unit-speed processors.

2. If each HI-criticality job executes for no more than its HI-criticality WCET and each LO-criticality job does not execute at all then each HI-criticality job can meet its (original) deadline *by beginning execution at or after its modified deadline.* This is

ensured by ensuring that the collection of "regular" (non-MC) tasks

$$\bigcup_{\chi_i=\text{HI}} \Big\{ (C_i(\text{HI}), T_i - \hat{T}_i) \Big\} \tag{6.6}$$

can be scheduled by Algorithm fpEDF to always meet all deadlines on the available m unit-speed processors.

According to the description above, the pre-processing phase can be specified in pseudo-code form in Figure 6.1. We provide an explanation of this pseudo-code below.

Step 1 checks to see whether Algorithm fpEDF can schedule the system if each LO-criticality job executes for up to its LO-criticality WCET, and each HI-criticality job executes for up to its HI-criticality WCET. If so, then the system can be scheduled directly by Algorithm fpEDF; else, Steps 2-3 are executed.

In **Step 2**, a minimum "scaling factor" x is determined, such that if all the HI-criticality tasks have their periods scaled by this factor x then the regular implicit-deadline task system obtained by combining these tasks with the LO-criticality tasks would be successfully scheduled by Algorithm fpEDF. The derivation of the value of x is as follows. According to Theorem 6.2, Algorithm fpEDF can schedule any task system with total utilization $\leq (m+1)/2$ (recall that m denotes the number of unit-speed processors). Since scaling the period of each HI-criticality task by a factor x is equivalent to inflating its utilization by a factor $1/x$, for ensuring LO-criticality schedulability by fpEDF we therefore need

$$\begin{aligned} & U_1(1)(\tau) + \frac{U_2(1)(\tau)}{x} \leq \frac{m+1}{2} \\ \Leftrightarrow & \frac{U_2(1)(\tau)}{x} \leq \frac{m+1}{2} - U_1(1)(\tau) \\ \Leftrightarrow & x \geq U_2(1)(\tau) / \Big(\frac{m+1}{2} - U_1(1)(\tau) \Big) \end{aligned}$$

This accounts for the first term in the "max". The second term is to ensure that scaling down the period of any HI-criticality task by this factor x does not result in the task having its LO-criticality WCET exceed its scaled-down period (equivalently, the term $\frac{C_i(\text{LO})}{xT_i}$ becoming > 1 for some HI-criticality task τ_i).

Task system $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on m processors.

1. **If** the regular task system
$$\bigcup_i \{(C_i(\chi_i), T_i)\}$$
is deemed schedulable on the m processors by Algorithm fpEDF, **then** declare success and **return**.

2. $x \leftarrow \max\Big(U_2(1)(\tau)/\big(\frac{m+1}{2} - U_1(1)(\tau)\big), \max_{\chi_i=\text{HI}}\big\{U_i(\text{LO})\big\}\Big)$

3. **If** the regular task system
$$\bigcup_{\chi_i=\text{HI}} \big\{(C_i(\text{HI}), (1-x)T_i)\big\}$$
is deemed schedulable on the m processors by Algorithm fpEDF, **then**

   $\hat{T}_i \leftarrow xT_i$ for each HI-criticality task τ_i

   declare success and **return**.

   else declare failure and **return**.

Figure 6.1: Global EDF-VD: The preprocessing phase.

Step 3 determines whether the HI-criticality tasks can be scheduled to meet all deadlines by Algorithm fpEDF once the behavior of the system transits to HI-criticality (i.e., after the time-instant t^* at which some job is identified to have executed for more than its LO-criticality WCET). If so, the modified deadline parameters — the $\hat{T}_i$'s — are computed.

In the end of this subsection, we state the following conclusions without proof, in order to provide a quantitative way to evaluate the performance of our global mixed-criticality scheduling method.

A following *sufficient schedulability condition* can be derived for our multiprocessor mixed-criticality scheduling algorithm:

Theorem 6.3. *Any task system τ satisfying*

$$U_1(1)(\tau) + \min\Big(U_2(2)(\tau), \frac{U_2(1)(\tau)}{1 - U_2(2)(\tau) \cdot 2/(m+1)}\Big) \leq \frac{m+1}{2} \tag{6.7}$$

is successfully scheduled by our algorithm on m preemptive unit-speed processors.

A speedup factor bound is computed for the global EDF-VD scheduling algorithm:

Theorem 6.4. *The processor speedup factor of the global EDF-VD scheduling algorithm is no larger than* $(\sqrt{5}+1)$*. That is, any mixed-criticality task system that can be scheduled in a certifiably correct manner on* m *unit-speed processors by an optimal clairvoyant scheduling algorithm can be scheduled by our algorithm on* m *speed-*$\left(\sqrt{5}+1\right)$ *processors.*

6.2.2 Partitioned Mixed-Criticality Scheduling

The partitioned scheduling algorithm assigns each task to a dedicated processor, then schedule all the tasks by EDF-VD algorithm on each processor. Therefore, we do not need to discuss the scheduling policy here because it remains the same. We only describe the partitioning algorithm, which has two phases:

1. During the first phase each HI-criticality task is assigned to some processor while ensuring that the cumulative HI-criticality utilization assigned to each processor does not exceed 3/4.

2. During the second phase each LO-criticality task is assigned to some processor while ensuring that the cumulative LO-criticality utilization assigned to each processor also does not exceed 3/4.

Observe that by Theorem 4.4, such an assignment procedure ensures that each processor remains schedulable by EDF-VD. The algorithm reports failure if it fails to successfully assign every task.

Now we provide a detailed description of our partitioned mixed-criticality scheduling algorithm. Let τ denote the implicit-deadline sporadic task system that is to be partitioned amongst m processors. Let us assume that there are n tasks in τ, of which n_1 are HI-criticality tasks. Without loss of generality, assume that $\tau_1, \tau_2, \ldots, \tau_{n_1}$ are the HI-criticality tasks, and $\tau_{n_1+1}, \ldots, \tau_n$ the LO-criticality ones. Let $\pi_1, \pi_2, \ldots, \pi_m$ denote the m processors. Let us suppose that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have been successfully assigned. We now explain how the task τ_i is assigned to a processor.

For any processor π_k, let $\tau(\pi_k)$ denote the tasks from amongst $\tau_1, \tau_2, \ldots, \tau_{i-1}$ that have already been assigned to it. Our algorithm assigns the task τ_i to any processor π_k satisfying

the following condition. If $i \leq n_1$ (i.e., if τ_i is a HI-criticality task) then

$$\left(U_i(\text{HI}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{HI}) \right) \leq \frac{3}{4} \tag{6.8}$$

else (i.e., $i > n_1$ and τ_i is hence a LO-criticality task)

$$\left(U_i(\text{LO}) + \sum_{\tau_j \in \tau(\pi_k)} U_j(\text{LO}) \right) \leq \frac{3}{4} \tag{6.9}$$

If no such π_k exists, then the algorithm declares failure: it is unable to partition τ upon the m-processor platform.

It can be easily proved that our algorithm provides a valid partition, which means after the processor assignment, all processors are schedulable by EDF-VD after all tasks in τ have been successfully assigned. We will not discuss the detailed proof here.

Also, a speedup factor bound is also calculated for the partitioned EDF-VD scheduling algorithm:

Theorem 6.5. *The speedup bound of the partitioned EDF-VD scheduling algorhim on an m-processor platform is $(8m - 4)/3m$, which asymptotically approaching $8/3$ as $m \to \infty$.*

The partitioning algorithm appears to be better, from the perspective of speedup bounds, when compared to the global algorithm. However, because neither of these two speedup factors are proved to be tight, we will not claim the supremacy of the partitioning scheduling algorithm.

CHAPTER 7

Conclusion

7.1 A Summary of Research Results

Due to the rapid increase in the complexity and diversity of functionalities that are performed by safety-critical embedded systems, the cost and complexity of obtaining certification for such systems is fast becoming a serious concern. We believe that in mixed-criticality systems, these certification considerations give rise to fundamental new resource allocation and scheduling challenges which are not adequately addressed by conventional real-time scheduling theory. In this dissertation, we focus on a mixed-criticality model that is particularly designed for representing mixed-criticality workloads. We derive several scheduling algorithms, including OCBP algorithm for scheduling mixed-criticality jobs, and EDF-VD algorithm for scheduling implicit-deadline and arbitrary-deadline mixed-criticality sporadic tasks. We conduct a thorough investigation of the schedulability properties of these algorithms, including quantitative load bounds on these scheduling algorithms, and quantitative performance guarantees according to the speedup factors.

OCBP algorithm is an ideal scheduling algorithm for mixed-criticality jobs. It is very efficient in the sense of both run-time complexities and computational resource utilizations. Based on priorities, the OCBP schedulability test and the OCBP run-time dispatcher are both easy to implement. We have shown the speedup factors of OCBP algorithm for arbitrary numbers of criticality levels. We also show that the speedup factors are lower (hence better) than the traditional WCR-scheduling algorithm. It is very impressive to note that OCBP algorithm has the best speedup factors in all fixed-job-priority scheduling policies, and more impressively, the best speedup factor 1.618 in all on-line scheduling policies for two criticality levels. It keeps an open question how good OCBP algorithm is if compared to an

optimal (and provably in exponential time assuming P≠NP) on-line scheduling algorithm. In order to answer that question, we need more exploration to discover the essence of optimal on-line scheduling algorithms. Current results only show the comparison of OCBP algorithm with clairvoyant algorithms which can assure that no computational resource is wasted but is impractical in reality. Also, it keeps an open question what is the best possible speedup factors in all on-line scheduling policies for arbitrary numbers of criticality levels, and consequently, whether there is an algorithm that achieves these factors.

EDF-VD algorithm is also an ideal scheduling algorithm for dual-criticality implicit-deadline sporadic tasks. The criteria of selecting jobs for execution (seeking the job with the earliest virtual deadline) is very effective for indefinite job releases. We also show that EDF-VD algorithm has a speedup factor of 4/3, which implies that EDF-VD algorithm always utilizes no less than 75% of the computational resources. Also, this is the best speedup factors in all online scheduling policies for two criticality levels and implicit-deadline tasks. It is not surprising to see that implicit-deadline mixed-criticality tasks can be scheduled with less resource wastes than mixed-criticality jobs and arbitrary-deadline tasks, because the conciseness of implicit-deadline tasks brings in more predictability — the same as in the non-mixed-criticality case. In fact, the hardness of scheduling mixed-criticality implicit-deadline tasks remains an open problem since the NP-hardness proof of MC-schedulability problem (Baruah et al., 2010c, 2012b) can not be applied to implicit-deadline case.

The choice of algorithm for scheduling mixed-criticality arbitrary-deadline sporadic tasks is more difficult to make. Both OCBP algorithm and EDF-VD algorithm can be extended to the arbitrary-deadline case. The OCBP algorithm for sporadic tasks keeps the speedup factors of OCBP algorithm, including the optimal speedup factor 1.618 for dual-criticality case. However, because the scheduling policy needs to alter the priorities of the jobs in run-time, the run-time complexity appears high. On the other hand, the EDF-VD algorithm for sporadic tasks keeps the low run-time complexity of the scheduling policy. As a trade-off, the computational resource waste is higher than OCBP algorithm, and so far only the dual-criticality case is solved by EDF-VD algorithm.

The multiprocessor mixed-criticality scheduling algorithms in this dissertation are designed based on EDF-VD algorithm for dual-criticality implicit-deadline sporadic systems.

As an initial effort, we show that this problem can be solved in both global and partitioned ways. The speedup factors, 3.236 for global mixed-criticality scheduling and 2.67 for partitioned mixed-criticality scheduling are shown to be neither optimal nor tight. Therefore, we expect to provide some intuitions to work on and we expect to see improved results based on our current progress.

7.2 Future Plan

Mixed-criticality scheduling research will be foreseeably attractive for a long time. After answering the fundamental questions in mixed-criticality scheduling theory, we can still perceive the vast blank in this area. Firstly, it is noticeable that the scheduling policies for recurrent tasks on uniprocessor platforms are swaying among approximation ratio (OCBP algorithm for sporadic tasks) and run-time complexity (EDF-VD algorithm for sporadic tasks), while the comprehensive methods that excel at both are limited to two criticality levels and implicit-deadline cases(EDF-VD algorithm for implicit-deadline sporadic tasks). The existence of the supreme uniprocessor scheduling policy remains an open question. Secondly, the scheduling theory on multiprocessor platforms is still incomplete because of the lack of optimality proofs and the restriction of task types, processor types and migration rules. Further research is required for the boundary of the performance of multiprocessor mixed-criticality scheduling algorithms for various task types (implicit-/arbitrary-deadline) on various processor types (identical/heterogeneous/unrelated) with various migration rules (unrestricted/job level/task level). Finally, the scheduling theory with dependency constraints (task dependency/resource sharing/hierarchical scheduling) is undeveloped. I would like to work on these problems and complete the mixed-criticality scheduling theory.

Besides the scheduling theory which only gives an abstract overview of the real-time systems, practice is always needed so that complex details of the real-world systems would be covered. The theoretical guarantee of the temporal correctness requires a flawless interaction between the schedulers and the other parts of the safety-critical embedded systems. In order to completely deliver the theoretical guarantees, it is necessary to look into the role of schedulers in such systems by studying the design and implementation of these systems, and

provide applicable scheduling components so that the temporal correctness will be seamlessly integrated in these systems. To meet this goal, issues like system architectures, component communications and run-time environments must be considered, and the awareness of possible failures must be established. We shall never be too confident in how to build accurate models and how to reasonably apply theoretical results to the practical systems. The importance of the connection between theory and practice shall never be underestimated.

BIBLIOGRAPHY

Audsley, N. C. (1991). Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, England.

Audsley, N. C. (1993). *Flexible Scheduling in Hard-Real-Time Systems*. PhD thesis, Department of Computer Science, University of York.

Baker, T. and Baruah, S. (2008). Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Dublin. IEEE Computer Society Press.

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

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2012a). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy). IEEE Computer Society.

Baruah, S., Bonifaci, V., D'Angelo, G., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2011a). Mixed-criticality scheduling of sporadic task systems. In *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 555–566, Saarbrucken, Germany. Springer-Verlag.

Baruah, S. and Burns, A. (2006). Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 159–168, Rio de Janeiro. IEEE Computer Society Press.

Baruah, S., Burns, A., and Davis, R. (2011b). Response-time analysis for mixed criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria. IEEE Computer Society Press.

Baruah, S. and Fohler, G. (2011). Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria. IEEE Computer Society Press.

Baruah, S. and Haritsa, J. (1997). Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, 46(9):1034–1039.

Baruah, S., Li, H., and Stougie, L. (2010a). Mixed-criticality scheduling: improved resource-augmentation results. In *Proceedings of the ICSA International Conference on Computers and their Applications (CATA)*. IEEE.

Baruah, S., Li, H., and Stougie, L. (2010b). Towards the design of certifiable mixed-criticality systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. IEEE.

Baruah, S., Mok, A., and Rosier, L. (1990). Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida. IEEE Computer Society Press.

Baruah, S. and Vestal, S. (2008). Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Prague, Czech Republic. IEEE Computer Society Press.

Baruah, S. K., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L. (2010c). Scheduling real-time mixed-criticality jobs. In Hlinený, P. and Kucera, A., editors, *Proceedings of the 35th International Symposium on the Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer.

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

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms.* MIT Press, third edition.

de Niz, D., Lakshmanan, K., and Rajkumar, R. R. (2009). On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the Real-Time Systems Symposium*, pages 291–300, Washington, DC. IEEE Computer Society Press.

Dorin, F., Richard, P., Richard, M., and Goossens, J. (2010). Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems.*

Eisenbrand, F. and Rothvoß, T. (2010). EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms.*

Ekberg, P. and Yi, W. (2012). Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy). IEEE Computer Society.

George, L., Rivierre, N., and Spuri, M. (1996). Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA: Institut National de Recherche en Informatique et en Automatique.

Ghazalie, T. M. and Baker, T. (1995). Aperiodic servers in a deadline scheduling environment. *Real-Time Systems: The International Journal of Time-Critical Computing*, 9.

Guan, N., Ekberg, P., Stigge, M., and Yi, W. (2011). Effective and efficient scheduling for certifiable mixed criticality sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria. IEEE Computer Society Press.

Herman, J., Kenna, C., Mollison, M., Anderson, J., and Johnson, D. (2012). Rtos support for multicore mixed-criticality systems. In *Proceedings of the 2012 IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '12, Beijing (China). IEEE Computer Society.

Horn, W. (1974). Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185.

Koren, G. and Shasha, D. (2003). Scheduling overloaded real-time systems with competitive/-worst case guarantees. In Leung, J. Y.-T., editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press LLC.

Lakshmanan, K., de Niz, D., Rajkumar, R. R., and Moreno, G. (2010). Resource allocation in distributed mixed-criticality cyber-physical systems. In *Proceedings of the 30th International Conference of Distributed Computing Systems*. IEEE Computer Society Press.

Lawler, E. L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546.

Li, H. and Baruah, S. (2010a). An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proceedings of the Real-Time Systems Symposium*, pages 183–192, San Diego, CA. IEEE Computer Society Press.

Li, H. and Baruah, S. (2010b). Load-based schedulability analysis of certifiable mixed-criticality systems. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, Scottsdale, AZ. IEEE Computer Society Press.

Li, H. and Baruah, S. (2012). Global mixed-criticality scheduling on multiprocessors. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy). IEEE Computer Society.

Liu, C. and Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61.

Mok, A. (1988). Task management techniques for enforcing ED scheduling on a periodic task set. In *Proc. 5th IEEE Workshop on Real-Time Software and Operating Systems*, pages 42–46, Washington D.C.

Mollison, M., Erickson, J., Anderson, J., Baruah, S., and Scoredos, J. (2010). Mixed-criticality real-time scheduling for multicore systems. In *Proceedings of the IEEE International Conference on Embedded Systems and Software*, Bradford, UK. IEEE Computer Society Press.

Park, T. and Kim, S. (2011). Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *Proceedings of the 11th International Conference on Embedded Software*, EMSOFT-2011, pages 253–262.

Pathan, R. (2012). Schedulability analysis of mixed-criticality systems on multiprocessors. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy). IEEE Computer Society.

Pellizzoni, R., Meredith, P., Nam, M. Y., Sun, M., Caccamo, M., and Sha, L. (2009). Handling mixed criticality in SoC-based real-time embedded systems. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, Grenoble, France. IEEE Computer Society Press.

Phan, L., Chakraborty, S., and Lee, I. (2009). Timing analysis of mixed time/event-triggered multi-mode systems. In *Proceedings of the 2009 Real-Time Systems Symposium, RTSS 2009. 30th IEEE*, pages 271–280. IEEE.

Phan, L., Lee, I., and Sokolsky, O. (2011). A semantic framework for mode change protocols. In *Proceedings of the 2011 Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 91–100. IEEE.

Real, J. and Crespo, A. (2004). Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26:161–197.

Ripoll, I., Crespo, A., and Mok, A. K. (1996). Improvement in feasibility testing for real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 11:19–39.

Santy, F., George, L., Thierry, P., and Goossens, J. (2012). Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy). IEEE Computer Society.

Shin, I. and Lee, I. (2004). Compositional real-time scheduling framework. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 57–67. IEEE Computer Society.

Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium*, pages 239–243, Tucson, AZ. IEEE Computer Society Press.

Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., and Sha, L. (2012). Memory access control in multiprocessor for real-time systems with mixed criticality. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy). IEEE Computer Society.