technische universität
dortmund

Masterarbeit

Approximated Timing Analysis in Real-Time Calculus

Vasco Fachin

November 2015

Gutachter:
Prof. Dr. Jian-Jia Chen
M.-Ing. Wen-Hung Kevin Huang

Technische Universität Dortmund
Fakultät für Informatik
Entwurfsautomatisierung für Eingebettete Systeme (LS-12)
http://ls12-www.cs.tu-dortmund.de

Abstract

Over the last decades, the increase of real-time and embedded system has lead to a more in depth analysis of this field of computer science. Traditional scheduling theory cannot be considered alone for the analysis of such complex systems. Moreover, distributed systems would need to verify the system performance as a whole: the analysis over the single components may be too pessimistic. The cost of such systems has to be minimised, therefore it is not possible to over-allocate resources in order to compensate for a too pessimistic performance analysis.

For this reason, many verification and validation methods have been implemented and developed in the last years. One of them, the MPA-RTC Framework (Modular Performance Analysis with Real-Time Calculus) has gained particular attention both within academia and industry sectors.

Real-Time Calculus offers a powerful mathematical back-round for the design stage performance analysis of distributed systems. It uses the concept of curves in order to represent potentially every event and resource patterns, both as incoming and outgoing values, thus providing an intuitive modelling framework for complex distributed and embedded systems, subject to real-time constrains.

However, real-time calculus suffers from an exponential complexity of its operation. Even simple models need minutes to be analysed in the real-time calculus toolbox. Furthermore, in same cases the computation is even impossible due to memory overflow errors. In order to reduce its complexity, extensions of real-time calculus, such as finitary real-time calculus, have been recently introduced. Nevertheless, they do not solve all the problems.

In this work we present a different approach to reduce the complexity of real-time calculus in some particular cases. In fact, a fixed-priority component, described by real-time calculus as a top-down chain of greedy-processing components, can be efficiently modelled using traditional scheduling theory. We will show how to adapt the current state-of-the-art of traditional scheduling theory to the event models described by real-time calculus. Moreover, we will present a fully polynomial-time approximation scheme that efficiently substitutes some of the operations within real-time calculus. To conclude, we also present a first implementation of these concepts, providing an evaluation between the original real-time calculus toolbox and our framework.

Zusammenfassung

In den letzten Jahrzehnten hat die zunehmende Verbreitung von eingebetteten und Echtzeit-Systemen zu einer Erweiterung der Analysemethoden für solche Systeme in der Informatik geführt. Die Anwendung der traditionellen Scheduling-Theorie ist zur Analyse von derart komplexen Systemen häufig nicht mehr ausreichend. Darüber hinaus muss die Leistung eines verteilten Systems als Ganzes analysiert und bewertet werden, denn eine getrennte Analyse der einzelnen Komponenten könnte zu pessimistische Ergebnisse liefern. Darüber hinaus müssen die Produktionskosten für solche Systeme minimiert werden, was es unmöglich macht, überschüssige Ressourcen zu allokieren, um eine Überabschätzung bei der Performance-Analyse zu kompensieren.

Aus diesem Grund wurden in den letzten Jahren viele Verifizierungs- und Validierungsverfahren entwickelt und implementiert. Das MPA-RTC-Framework (Modular Performance-Analyse mit Real-Time-Calculus) hat sowohl in der Forschung als auch in der Industrie besondere Aufmerksamkeit erlangt.

Real-Time Calculus bietet die Möglichkeit einer leistungsfähigen mathematischen Performance-Analyse, die bereits in der Entwicklungsphase von verteilten Systemen eingesetzt werden kann. Es nutzt das Konzept von Kurven, um jedes potenziell mögliche Ereignismuster sowie die zur Verfügung stehenden Ressourcen zu repräsentieren, wodurch ein intuitives Modellierungs-Framework für komplexe verteilte eingebettete Systeme mit Echtzeitanforderungen entsteht.

Ein Problem von Real-Time Calculus ist allerdings, dass es bei der Anwendung über eine exponentielle Komplexität verfügt. Selbst einfache Modelle benötigen Minuten, um in der Real-Time Calculus Toolbox analysiert zu werden. Weiterhin wird in bestimmten Fällen die Berechnung aufgrund eines Speicherüberlaufs sogar unmöglich. Um die Komplexität zu reduzieren, wurden in den letzten Jahren verschiedene Erweiterungen - wie beispielsweise *finitary Real-Time Calculus* - vorgestellt. Nichtsdestotrotz werden durch diese Erweiterungen die Probleme nicht vollständig gelöst.

In dieser Arbeit wird ein neuartiger Ansatz vorgestellt, um die Komplexität der Real-Time Calculus-Analyse für einige Anwendungsfälle zu reduzieren. Beispielsweise wird eine *Fixed-Priority*-Komponente in Real-Time Calculus als eine *Top-Down*-Kette von sogenannten *Greedy Processing Components* (GPC) dargestellt. Eine solche Verkettung kann mit

Hilfe der traditionellen Scheduling-Theorie effizient modelliert und analysiert werden. Es wird gezeigt, wie die State of the Art der traditionellen Scheduling-Theorie auf die von Real-Time Calculus beschriebenen Ereignismodelle angepasst werden kann. Darüber hinaus wird ein Voll-Polynomialzeit-Approximationsschema (FPTAS) präsentiert, das einige von den in Real-Time Calculus angewendeten Operationen durch effizientere Operationen ersetzen kann. Abschießend wird eine erste Implementierung dieser Konzepte vorgestellt und eine vergleichende Analyse zwischen der ursprünglichen Real-Time Calculus Toolbox und dem vorgestellten Framework durchgeführt. german

A Ercole

Acknowledgements

A no è mai facil lasciâ cjasa sô e lâ ator par il mont. Soradut in paîs na che la lenga ca tabain a è cusì dificil come il todesc. Propit par chest a si è un tic plui orgoglious quant ca si riva a concludi alc di cusì dificil come il studi.

E i no sares mai rivât a fa dut chest, se no fos stât par l'Ercole. Dut chest lavôr di tesi a l'è inprin dedicât a lui.

A Giulia a no coventin ditas paraulas, d'altronde ce podia mai dî di biel un cjargnelat, nou che no podin disi nua di plui che "ti voi bon". Ma sigûr a sa beniscim che chest lavôr a l'è in grant part merit encje so, ca mi a iudât ducj in chest agns, encje cuant chi eri pi a tor che a studiâ.

Encje a Maria Teresa a l'è dedicât dut chest. Ca nu si sopuarta cuant che sin fûr di cjasa e a si parecja duta la tradota che portin in Gjermania, savint ben che chest mucs a no è cai vadin trop.

E infin, a me. Parce che chest lavôr al conclud chest gno percors di studis. Tacât masa timp fa, ma forcit finît tal moment iust. E ala fin nome io i sai ce tant ca mi è costât e la fadia chi ai fat. Si. Chesta laurea a è propit par me.

Now back to the common language, I would like to thank Professor Chen and M.Ing. Wen-Hung:

真心感你一直以的支持和鼓

Contents

Chapter 1

Introduction

The last decades, real-time systems have become ubiquitous in everyday situations. From smart-phones, to in-car multimedia, and up to the Airbus AFDX Network system, these all need to work under certain constrains, i.e. bus bandwidth, CPU speed, whilst delivering the computation results within a predefined amount of time.

As an intuitive example for real-time system, we can look at a multimedia application, such as a video-streaming device. In this situation the CPU has to process the video file and deliver the correct pixel matrix to a display. If the CPU fails to deliver a frame at a certain point of the video reproduction, this frame becomes useless, even if it had been processed correctly.

In particular case, the failure to deliver the result on time does not have a strong impact on the end-users: they will only miss one frame and probably barely notice it. However, such a failure to deliver the result on time could lead to catastrophic consequences in some applications: if the air-bag control unit fails to process an event sent by the sensor which recognise an incident, this could lead to the loss of a human life[1].

1.1 (Embedded) Real-Time Systems

From the above examples, it is immediately clear that a real-time system has different characteristics than a general-purpose system such as a desktop computer. These properties are well summarised by Buttazzo in [8], of which we give here a quick overview:

Timeliness: the computed result has to be delivered within a time frame, also called *deadline*. The correctness of the computation does not depend only on the result but also on the time at which this is delivered. The time constrains are usually divided in three categorise: *hard*, if violating the timeliness produces a catastrophic consequence; *firm*, if the result not delivered on time is useless; and *soft*, if the result looses its value after missing the deadline.

[1] See also the Patriot missile incident reported in [8] p.3.

Predictability: given the importance of its timeliness, a real-time system needs to be predictable in ideally every situation. The tasks and properties of the system should be known *a-priori* in order to analyse them offline before deploying the system.

Reliability: if one of the components fails - both hardware (HW) or software (SW) -, the system should be able to operate further without crashing completely.

Efficiency: most real-time systems are embedded in small appliances. It is therefore often needed to take into account memory space, energy efficiency, code size, and other constrains.

Robustness: real-time systems are designed for the worst case scenario. Overloads and peaks need to be considered at the system design stage.

The definitions presented above clearly show that the predictability of a real-time system is extremely important when analysing it. The ideal situation would be to give a 100% guarantee over the system's life time [36], though it is impractical, if not impossible, to do so for more complex systems.

The most important method in order to achieve predictability is the *scheduling policy*. In this way, with an offline analysis of the system's characteristics, we can derive - if possible - a feasible schedule of the system tasks, so that all of them will meet their deadlines.

The predictability of a system depends on various factors: from the CPU properties up to the programming language characteristics, all of these components introduce some sort of *non-determinism*, which affects the derivation of an exact *worst-case execution time* (WCET) needed to adopt the correct scheduling policy.

Furthermore, many real-time systems are embedded in some control units. Each of these units can cover a wide range of different components, architectures, and tasks that have to be processed, making a case generalisation impossible. The components may also need to communicate with each other: one processed task may be the incoming task of the next component. In such a so called *distributed system*, it is clearly impossible to derive the overall predictability by analysing each task or component as a single entity.

1.1.1 Simulation and Formal Methods

For this reason, evaluation and validation methods have been widely used in order to verify if the designed system will satisfy the given requirements. There are mainly two different approaches to this problem:

Simulation is nowadays widely used in the embedded system industry for performance verification. Tools like System C, VHDL and Matlab/Simulink have been largely extended in order to give a higher abstraction level and modularity in the modelling process of a given system. Furthermore, as these tools are more and more used in

the industry design phases, standard components of embedded systems have already been implemented and thus can be reused[42]. Moreover, simulations can potentiality represent any given system with complex and dynamic interactions on the same [28].

However, the hardware-software simulation process is typically computationally complex and running time execution is often long. The model construction is not a trivial task and the result might also not be detailed enough to meet the system requirements. In addition, simulation is a trace-based method. This means, that in order to determine, for example, a worst-case execution time, the designer has to provide the model with an input trace that will cover this scenario. In general, a performance analysis on every possible situation over a model is not feasible due to the state-space explosion. In fact, if we consider all the possible inputs combinations, the cases needed to be evaluated grow very rapidly, thus making a full analysis impossible.

Formal Methods of performance analysis uses a different approach. The core of this technique is the utilisation of a formal method, i.e a mathematical tool, that allows a higher abstraction level w.r.t. simulation. The main advantage is that we are able to derive hard or even exact bounds regarding the performance of the system. Moreover, the higher abstraction level can simplify the representation of the system without altering its behaviour. However, this methods can hardly be automated, that means that the model of the system has to be done by hand, since errors in the implementation could lead to a false result. Furthermore, only basic systems can be analysed since more complex ones lead to a state-space explosion.

Some authors, such as Perathoner in [28], describe an additional category of validation methods: the state-based verification, such as for example the timed-automata [18]. The advantage of this methods is that they derive an exact boundary of system performance. However, the state-space explosion "*severely inhibits their practical application*" [28, p.9], meaning that for large and complex system, the space-exploration grows exponentially, thus implicating long execution times for the validation.

1.1.2 Modular Performance Analysis with Real-Time Calculus

Real-time calculus (RTC) represents a formal method for performance analysis that has become well-known amongst the scientific community. This mathematical framework is embedded into the so-called *Modular Performance Analysis* (MPA), which defines an analytical method for the performance evaluation of distributed systems [28].

The advantage of MPA-RTC is that it does not consider tasks like in the classical real-time scheduling analysis, but works instead with event and resource streams. The idea is to model an incoming event stream as a pair of curves defined over a time interval, and not over time: the curves will then represent the upper and lower bound of incoming

events within a time interval. The same methodology is used to represent a resource stream. This pair of curves are then used as input elements for an *abstract component*, which can model a CPU or a bus system. The function of this abstract component is to derive the timing properties of the analysed system and to calculate the outgoing event and remaining resource curves. In real-time calculus this is mainly done by using a so-called greedy processing component (GPC), which acts in a greedy fashion for a first-in first-out buffer under the resource availability condition[41]. Through this component it is possible to determine the maximum delay, i.e. the amount of time that passes from the task arrival to its completed execution, and the maximum buffer of a task, i.e. the maximum amount of events that will be stored in the first-in first-out (FIFO) queue.

It is clear to see that with such a representation, we can easily interconnect more components in order to model a distributed system. In fact, one of the main advantages of MPA-RTC is the possibility of analysing end-to-end delays of multi-component systems.

However, even though MPA-RTC offers a high abstraction level for formal verification, it suffers from a huge drawback. Since the curves are unlimited for definition, it is impossible to represent them exactly. Instead the MATLAB®toolbox for real-time calculus uses a segment representation for the curves, with an aperiodic and a periodic part. The complexity of real-time calculus depends then on the number of segments used to represent the curves, yet even a simple plus operation augments exponentially the number of these segments.

1.2 Motivation

The high complexity of real-time calculus has been the object of academic research in the last few years, see the works [38] and [41]. In 2013, Guan et al. in [17], proposed an extension called *finitary* real-time calculus in order to reduce the overall complexity. The idea is to truncate the curve after a particular point while still obtaining the same precision as in the original framework. However, though reducing the complexity, the finitary real-time calculus is still of pseudo-polynomial complexity.

The motivation of this work is to reduce, in some particular cases, the overall complexity of real-time calculus by adding a "controlled" error in its curve representation. Real-time calculus defines a *fixed-priority* component [42], which is depicted in figure 1.1. This is done by hierarchically interconnecting in a top-down fashion a number of GPC components for the given task numbers. However, this method is practically inefficient, since computing the maximum delay of the lowest priority tasks one has to derive the remaining resource curves after every GPC component.

This work aims to implement a new component within real-time calculus, which uses classical scheduling analysis in order to examine a set of tasks that have to be scheduled in a fixed-priority fashion. This component can calculate the delay of the single tasks, whilst not

computing every remaining resource curve except for the last, which is represented in figure 1.2. Ideally the component would also derive the outgoing events curves, nevertheless this would require a more in-depth study of both curves representations and classical scheduling theory.

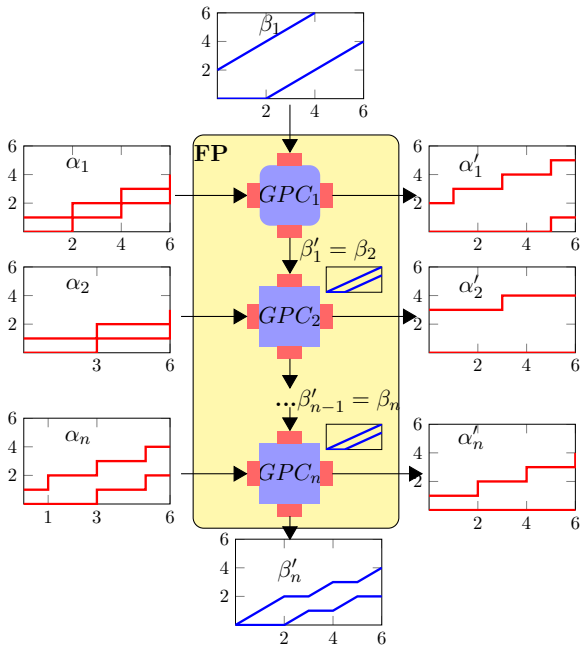

Figure 1.1: Fixed-Priority Component as in real-time calculus

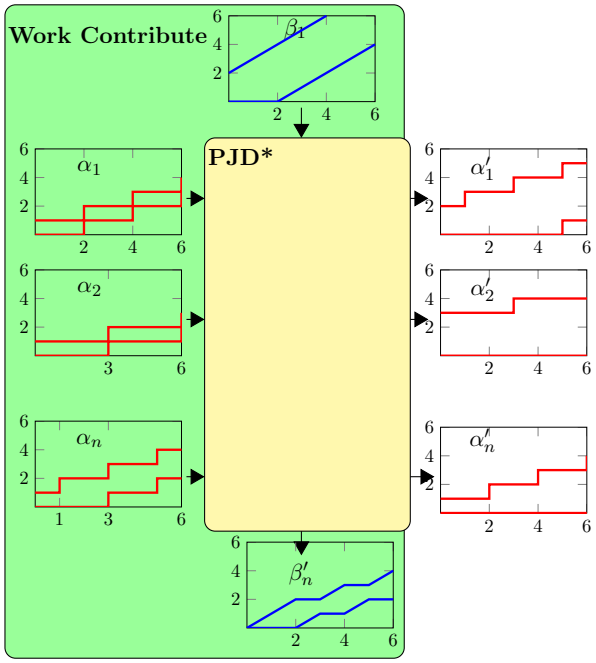

Figure 1.2: PJD* Component as derived with this work

1.3 Work Structure

In this chapter we presented the concepts handled through this work. Real-time system and schedulability analysis were introduced, as well as simulation and formal methods for system validation.

Real-time calculus plays a central role in this work, for this reason chapter 2 will be dedicated to formally define its basic concepts. The abstraction level given by the curve representation of concrete streams is described in this chapter. Moreover we introduce the different event models that are more common within this framework, i.e. periodic tasks, periodic tasks with jitter and periodic tasks with burst. The last section of the chapter will introduce the performance analysis components that are used in real-time calculus, whilst maintaining the focus on the concrete components.

Chapter 3 firstly introduces the min-plus max-plus algebræ, which are both an advantage and a drawback of RTC. The high complexity of the framework, based on the operation on infinite curves, is here reported. Furthermore, we will provide an overview of the related work of the complexity reduction methods for real-time calculus: first the curve representation will be defined and an then the so-called finitary real-time calculus will be analysed. In addition, an overview of the current approximation methods used in real-time calculus will be given.

In chapter 4 we introduce the main concepts of classical scheduling theory, which we used in order to implement a new component for real-time calculus. We will concentrate on fixed-priority algorithms and subsequently introduce the theory of polynomial time approximation schemes and their application in scheduling theory.

Chapters 5 and 6 represent the main contribution of this work, with the former formally defining the idea and the algorithm implemented in the latter, with the evaluation of the new component. In chapter 5 we introduce the concepts developed under the classical scheduling theory. At first we refine the idea of a new $PJD*$ component for real-time calculus, then we extend the current work in the field of approximate scheduling analysis in order to bind it to the real-time calculus framework.

In chapter 6 we present the implementation of the algorithm, which has been done using MATLAB®. The decision is based on the fact that the original RTC-Framework [40] has been implemented in the same programming language, so that our implementation and the original can be directly compared.

Chapter 7 concludes this work by underlying open problems and possible future developments.

1.4 Related Work

The central research of this work is of course based on real-time calculus. In the year 2000, Thiele et al. [39] presented their framework, outlining how it would be possible to model discrete events systems without recurring to classical scheduling theory[2]. Inspired by the network calculus theory by Boudec et al. [7] extending the work of Cruz in [12], real-time time calculus is based on the min-plus/max-plus algebra, which has a wide literature available, see for example Schwiegelshohn and Thiele in [35], Baccelli et al. in [5] and Cuninghame-Greene in [13].

Consequently real-time calculus has become popular within the scientific community for its easy to understand semantic and its high abstraction level, which allow a straightforward implementation of a distributed system. Focusing on the timing properties of a model, RTC has also been widely studied for its possible interconnection with other formal methods. For example, Phan et al. in [30] presented a *Multi-Mode Real-Time Calculus* which aims to solve the problem of the statelessness of the RTC. The idea is to try to define a state-based model such as in the timed automata method specified in [3] and [18]in order to represent states in real-time calculus as well. In addition, Santinelli et al. in [34] extend the framework to the domain of probabilistic real-time systems.

However, due mainly to the widespreadness of RTC, its drawbacks have also been analysed. In an internal report, Suppinger et al. [38] firstly remark how the curve must not be analysed in its entirety, but only a part of it can suffice. However, this point was given as an input. Later, Guan and Yi in [17], in their finitary real-time calculus work, formally defined the concept of *maximum busy window slice* to successfully analyse a pair of curves by truncating them after a mathematically determined point. Nevertheless the complexity the finitary RTC still remains pseudo-polynomial.

Additionally, some studies on real-time calculus focused on the possibility of approximating the curves. Wandeler gives in [41] a possibility of a linear approximation of the curves by a single linear function, which was however way too pessimistic. Chakraborty et al. in [9] proposed a three-pieces approximation for the curves, but did not report any evaluation on the same. Later Albers et al. in [1] and [2] analysed this approach and concluded that it is not suitable for complex systems.

For this reason, we turned our attention to the enormous literature in the field of classical scheduling theory. The well-know results of Liu and Layland in [22] caught our attention for the domain of fixed-priority scheduling algorithms. These algorithms, as mentioned above, are modelled by real-time calculus by interconnecting GPCs components. Furthermore, aside from the rate-monotonic scheduling scheme, the deadline-monotonic priority

[2]To be precise, a first glimpse of the work was firstly introduce on the internal journal of the TIK department at the ETH Zürich by Naedele, Thiele and Eisenring, although it was than named *Variable Task Model*. See [24]

assignment reported by Lehoczky in [21] showed a necessary and sufficient feasibility test for periodic tasks with deadlines smaller or equal to their periods. Their approach used the so-called workload analysis in order to derived and exact schedulability test. From these works, various papers extended this field during the last 30 years, among which Audsley et al in [4] proposed a deadline-monotonic scheduling theory, with sufficient - and necessary and sufficient tests, which points out the period-related complexity.

From this work, Fisher and Baruah opened the approximation algorithms theory for fixed-priority tasks [15]. At first the authors restricted their attention to a feasibility test for task with constrain deadline, but they soon extended it to arbitrary deadlines [14]. Nevertheless, later research done by Nguyen et al. [25] showed that the approximation algorithm for arbitrary deadline tasks was not correct and proposed a modification of the same. The key point of this particular task model is that more than one job of a task can be present at the same time, hence leading to a more complex analysis.

Furthermore, the focus on approximation algorithms moved to the workload analysis, since deriving the worst-case response time of a tasks will give an exact schedulability test. Richard et al. in [31] extended the algorithm of Fisher and Baruah both within the response time analysis and the periodic tasks with jitter domain. Later, in 2014, Nguyen et al. [26] continued this research to include a response time analysis for tasks with arbitrary deadlines.

Chapter 2

Real-Time Calculus

Over the last few years, increasing competition among industry has led to a progressive decrease of the time-to-market span. Since the early stages of the design phase, it is necessary to verify that the system can meet the desired requirements. A performance evaluation is essential at the very first steps of the system construction. The designer is faced with problems such as how much memory the system should have or the minimum processor speed needed in order for the end system to meet all the specified characteristics. Furthermore, due to the high demand of maximising the profit, i.e. avoiding unnecessary costs, it is not any more possible to over-allocate resources for the designed system [29].

This particular area of the development process takes the name of *system-level performance analysis* [41]. As already mentioned in the introduction, there are basically two ways to perform a system level analysis: simulation and formal methods.

The widespread utilisation of simulation methods such as System C or VHDL gives the designer the possibility of modelling basically every system. Due to a high modularisation, one can reuse some previously built components within a new model. However, a simulation method usually has a high computation complexity, making the process itself really slow. Moreover the simulation needs an input trace, which is usually difficult to represent, due to the fact that the state space of a model is vast. Furthermore, in order to calculate, for instance, the worst-case or best-case scenario, the designer has to feed the model with the appropriate trace. It is clear that with the increasing of the complexity of the system this becomes impossible. This leads to the so called *corner cases*, i.e. the situations that cannot be discover with the simulation process and are therefore left out of the performance analysis process (See figure 2.1).

Nevertheless, particularly in the field of *soft* real-time system, simulation is widely used. In this area, one looks more at the average case than at the worst or best case scenarios, making simulation tools suitable for purpose of analysis [28].

Formal methods for system-level performance analysis give the designer a much higher level of abstraction. The goal is not to try to find an approximation to the real worst-case

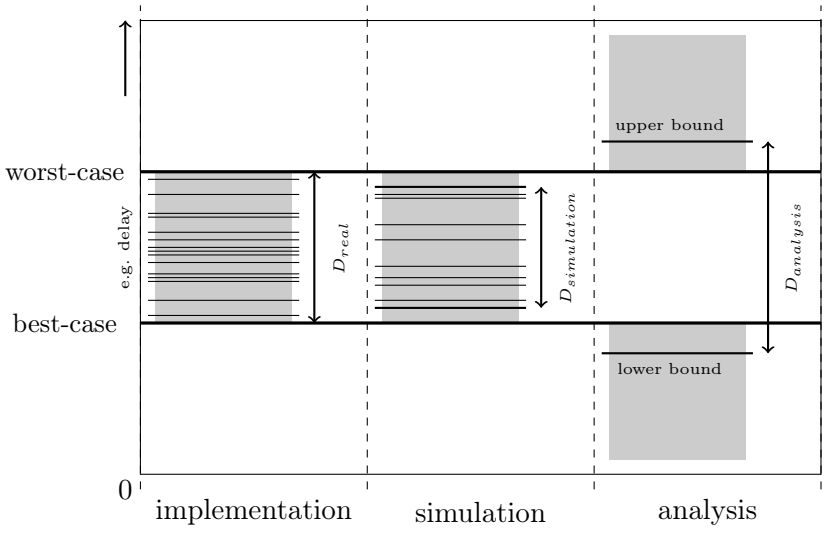

Figure 2.1: Comparison between the range of considered domain D: real implementation, simulation, and formal analysis [41].

or best-case, but to give an upper and lower bound of execution. If, for example, we consider the end-to-end delay of our system, the computed result using a formal method will always return either the exact value or a larger one. That means that, if successful, we can guarantee that our system will meet its requirements.

However, formal methods also have theirs drawbacks. Their modelling scope is fairly limited: usually the components we are able to represent are restricted due to the fact that is not always easy to reproduce their behaviour mathematically. Moreover the mathematical depiction of the system may also be not tight enough for our purposes - the so called *abstraction loss* -, leading to a possible over-estimation of the resources needed. Another important issue of formal methods is their implementation complexity: usually these methods are not that easy to understand and this process is, at the time of writing, difficult to automatise. For this reason the implemented model of a system could also not be detailed enough for our purposes or could even contain implementation errors since it has to be made by hand.

Nevertheless, the higher abstraction level and the possibility of analysing a system from its early design stages have made formal methods an interesting field of research over the last decade. Among the frameworks used for formal performance analysis, real-time calculus has known an increasing popularity since its first presentation. The main advantage of real-time calculus is that it can ideally represent every possible event model, due to the fact that its representation of tasks is done by means of curves based on real input traces. Moreover, the same concept is extended in order to represent different resource patterns, giving the framework a wide spectrum of implementation.

In this chapter we introduce real-time calculus formalisms and definitions. Our main focus will be on the representation of curves and the operations thereupon, as these characteristics have motivated this work. In order to better understand the semantic of real-time calculus, we introduce the concepts of convolution and deconvolution, which are based on the min-plus and max-plus algebræ, the core of this mathematical framework.

2.1 RTC - Basic

Real-time calculus has its basis in the Network Calculus (NC) [7], a theory of deterministic queuing for communication networks. RTC was first presented by Thiele et al. in [39] and aims to extend network calculus to the domain of real-time systems. To do so, RTC exploits the characteristics of events and resource streams in order to underline their timing properties. Instead of concentrating on some particular event model, RTC uses curves defined over a time interval to represent potentially every possible incoming stream.

2.1.1 Event Models

The first thing to notice about RTC is that instead of working within a time domain, it is defined over an interval domain. Using a *sliding-window* technique over a concrete trace of incoming events, we are interested in finding out how many events will at most or at least be delivered within this window, i.e. the interval. Doing so for every interval produces the curves used for the analysis.

Let $R(t)$ denote a *concrete* event stream within the time frame $[0, t)$. The $\alpha(\Delta)$ curve is a tuple $[\alpha^l(\Delta), \alpha^u\Delta]$ representing the *lower arrival curve* and *upper arrival curve* respectively, which are on the other hand *abstract* events models [42]. It is once again important to notice that the αs curve is defined over a time interval $\Delta > 0$, i.e. in every time interval of length Δ we will have at least $\alpha^l(\Delta)$ and at most $\alpha^u\Delta$ incoming events.

Formally stated, the relationship between concrete and abstract streams are given by the following definition:

$$\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s), \forall s < t \tag{2.1}$$

Although RTC has the expressiveness of representing potentially any event stream, it is sometimes easier to use a formal description depicting the event model. We give here a quick overview on the most common patterns used in real-time calculus.

Periodic Tasks The most simple task pattern is the periodic one. In this case tasks arrive every p units of time. The upper arrival curve depicts the behaviour of having an incoming event coinciding with the starting time of the interval we consider, whilst the lower arrival curve describes the situation of having "just missed" an event, so an event

happening first after p time units (See figure 2.2). A periodic task is formally described in terms of curves by the following equations:

$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta}{p} \right\rfloor, \alpha^u(\Delta) = \left\lceil \frac{\Delta}{p} \right\rceil \tag{2.2}$$

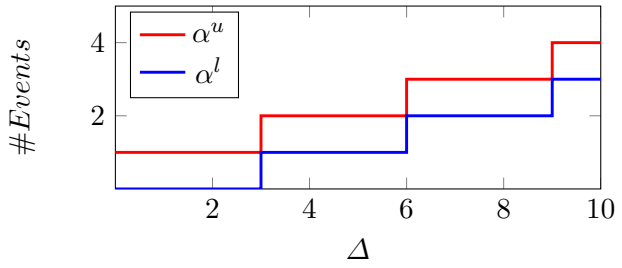

Figure 2.2: Arrival Curves for a period task with $p = 3$

Periodic Tasks with Jitter Another common pattern for task analysis is the one influenced by a *jitter*. A jitter is a deviance of the event arrival time that can be either delayed or anticipated with respect to the original period. In this behaviour the jitter is defined as $j \leq p$. Figure 2.3 represents the arrival curves for such a pattern. The upper curve depicts the case where all of the arrival times are anticipated by j time units, while the lower shows all the events to be delayed by j units. The upper and lower curves are defined by:

$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor, \alpha^u(\Delta) = \left\lceil \frac{\Delta + j}{p} \right\rceil \tag{2.3}$$

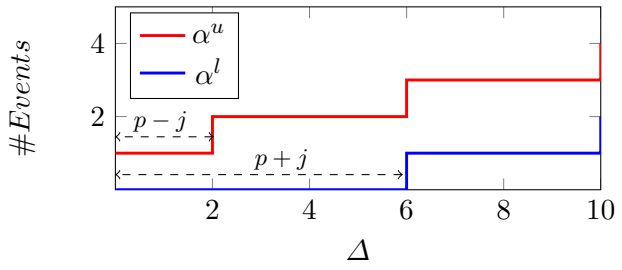

Figure 2.3: Arrival Curves for a task with period $p = 4$ and jitter $j = 2$

Periodic Tasks with Burst The last common pattern found in literature is a so called periodic task with burst, usually called a *pjd*. This particular model represents the condition where the jitter is not bounded any more by the period of the task, but the events still cannot overlap each other. This means that if we use the sliding window approach, we could also find conditions where we have an increased number of incoming events within the considered interval: this takes the name of *burst region* [32]. In order to bound this region, an addition parameter $d \geq 0$ is introduced, which depicts the minimum inter-arrival

time between two consecutive events (see Richter et al. in [33] and [32]). The distance can be seen as the impossibility of delivering more than one event within d by some sensor or CPU.

This model is widely used within the RTC domain. Its expressiveness is suitable to represent many concrete input traces, so that Künzli et al. in [19] derived an algorithm to transform ideally every event stream by means of a so called pjd model. On one hand, this representation reduces the computation complexity of the framework, but on the other it introduces the field of approximation in RTC. In fact, some error is necessarily introduced in the computation process by depicting a general arrival curve as a pjd model. The curve representation for the upper bound takes now the form of a minimum function - that is because the burst region is bounded by the minimum distance d:

$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor, \alpha^u(\Delta) = \min\left(\left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{d}{p} \right\rceil \right) \tag{2.4}$$

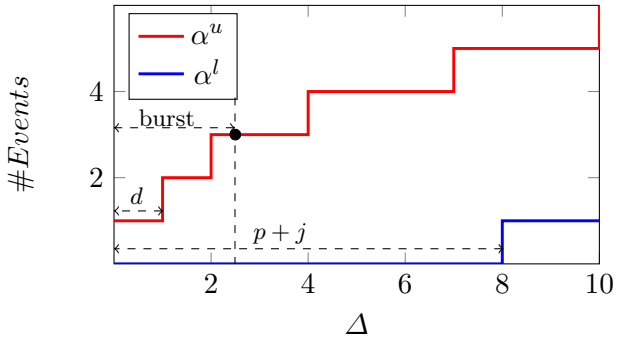

Figure 2.4: Arrival Curves for a task with period $p = 3$ and jitter $j = 5$ and minimum inter-arrival distance $d = 1$

2.1.2 Resource Models

Similarly, real-time calculus uses resource streams in order to represent elements of the system such as CPUs or buses. A concrete resource stream $C(t)$ characterises the resource availability within the time interval $[0, t)$. Again, we are more interested in an *abstract* resource model. These are defined in this case by the tuple $\beta(\Delta) = [\beta^l(\Delta), \beta^u(\Delta)]$ over any time interval of length $\Delta \geq 0$, which correspond to a so called *lower service curve* and *upper service curve* respectively. The relation between the resource stream and the service curve is stated by the following equation:

$$\beta^l(t - s) \leq R(t) - R(s) \leq \beta^u(t - s), \forall s < t \tag{2.5}$$

At this point is worth noticing that whilst the method of reproducing the event and resource stream appears to be the same, this is slightly different by means of the domain. In fact the arrival curves are defined for the number of events over a time interval, while

the service curves denote the number of cycles within the interval Δ. The difference can be stated by defining if the curve are *event based* or *resource based*. Usually in the RTC literature one can find the event based curves expressed by $\overline{\alpha}$. To obtain the resource based curve one can easily use a factor w characterising the number of cycles needed in order to process an event for a given task, i.e. $\alpha(\Delta) = w \cdot \overline{\alpha}(\Delta)$.

However, as Stoimenov in [37] stated, real-time calculus is also capable of handling different execution times for a task, i.e. two events of the same task can have distinct cycle requirements. This can be expressed by means of a so called *workload curve*. The workload curve is a tuple $\gamma(v) = [\gamma^l(v), \gamma^u(v)]$ denoting the minimum/maximum resource units needed to compute any v subsequently events. The relation between arrival, service, and workload curves is straightforward: if we represent all our curves as resource based, we obtain:

$$\alpha^l(\Delta) = \gamma^l(\overline{\alpha}^l(\Delta))$$

$$\alpha^u(\Delta) = \gamma^u(\overline{\alpha}^u(\Delta))$$

Even though the service curves in real-time calculus are able to theoretically express every resource model, some basic patterns have been formally described in order to simplify the comprehension of the framework.

Fully Available Resource A fully available resource is the easiest pattern we can implement. Let us take for example a CPU, whose cycles depend on the processor speed. In RTC these are expressed through the *bandwidth* parameter b, representing the available cycles per time unit. In figure 2.5 the bandwidth is set to one delivered cycle every one time unit. Formally this is stated by:

$$\beta^l(\Delta) = \beta^u(\Delta) = B \cdot \Delta \tag{2.6}$$

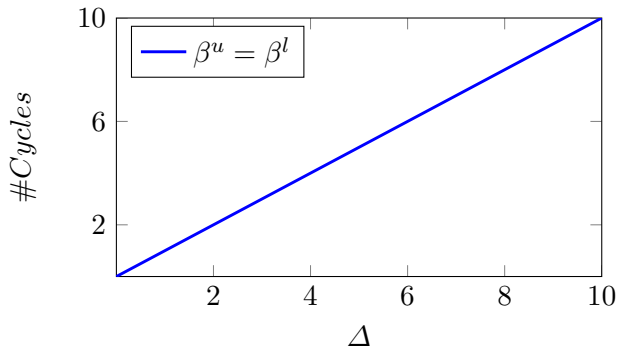

Figure 2.5: Service Curves for a resource with 100% availability, $B = 1$.

Bounded Delay Although widely used in the literature, a given resource may not be fully available in all situations. For example, a task may have to wait a certain amount of time BD, called *bounded delay*, while the resource is fully dedicated to another task [23]. In this case at any point in time the resource may not be available for the current task, whereas after the determined bound the bandwidth is still defined by B. The following equation gives its mathematical description.

$$\beta^l(\Delta) = B \cdot \Delta - BD, \beta^u(\Delta) = B \cdot \Delta \tag{2.7}$$

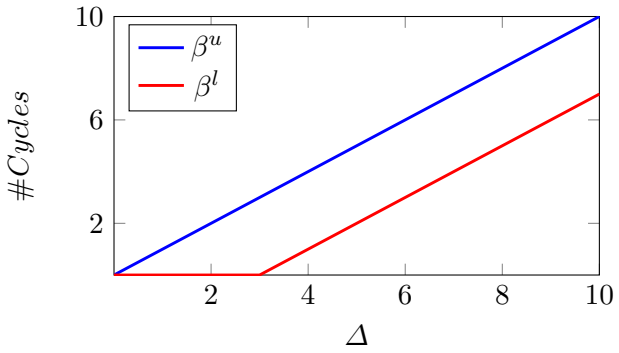

Figure 2.6: Service Curves for a resource with bounded delay $BD = 3$ and $B = 1$.

Time Division Multiple Access Another case of resource sharing is characterised by the so called *time division multiple access*. Within this pattern, the resource is divided into cycles of length c, each containing a number of slots s_i with a pre-given length. Each slot is assigned exclusively to a fixed task or task-set, so that the resource is allocated with bandwidth B for s_i units of time every c units, as expressed by the formulæ:

$$\beta^l(\Delta) = \left(\left\lfloor \frac{\Delta}{c} \cdot s + \min\left(\Delta \mod c, s\right) \right\rfloor \right) \cdot B \tag{2.8}$$

$$\beta^u(\Delta) = \left(\left\lfloor \frac{\Delta'}{c} \cdot s + \min\left(\Delta' \mod c, s\right) \right\rfloor \right) \cdot B \tag{2.9}$$

where $\Delta' = \max(\Delta - c + s, 0)$.

2.1.3 Concrete and Abstract Components

In an embedded system, the above mentioned input traces are typically processed by some component such as a CPU that will than produce an outgoing event trace for the computed tasks, thus consuming some processor cycles in order to do that.

This behaviour is represented in RTC by means of a concrete component. The R and C streams are processed through this component and it will derive the corresponding outgoing event stream R' and the remaining resource stream C'.

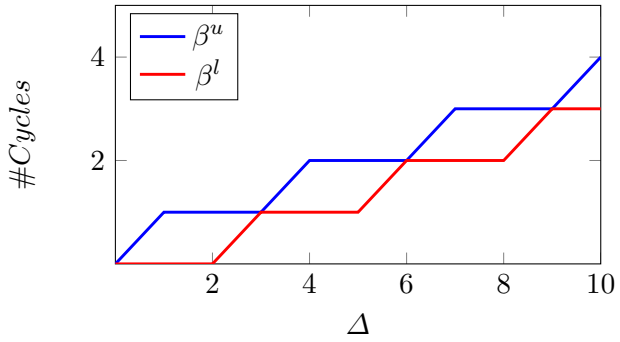

Figure 2.7: Service Curves for a a time division multiple access resource, with a $s = 1$ slot every $c = 3$ cycles and $B = 1$.

Again, since the exact concrete streams are not known during the design phase, we are more interested in considering an abstract representation of the component. The outgoing event curve is expressed by the tuple $\alpha'(\Delta) = [\alpha'^l(\Delta), \alpha'^u(\Delta)]$ and the remaining service curve by $\beta'(\Delta) = [\beta'^l(\Delta), \beta'^u(\Delta)]$. The resulting streams will depend on the semantic of the component and can be formally defined by the following equations:

$$\alpha' = f_\alpha(\alpha, \beta) \tag{2.10}$$

$$\beta' = f_\beta(\alpha, \beta) \tag{2.11}$$

One of the still open challenges of real-time calculus is indeed to formally define new semantics for equations 2.10 and 2.11. However one type of abstract component has become widely use in the RTC examples. This component takes the name of *greedy processing component* (GPC), whose characteristics have been summarised by [42] (see also figure 2.9):

- the component is triggered by incoming events;
- a fully preemptable task is instantiated at every event arrival to process the incoming event;
- active tasks are processed in a greedy fashion in FIFO order;
- the resources' availability restricts the computation.

2.2 Performance Analysis

In this section we introduce the mathematical description for analysing a GPC component. We start by reporting the backlog and delay definitions for a single component. After that, we present the semantic of a network of components in real-time calculus and

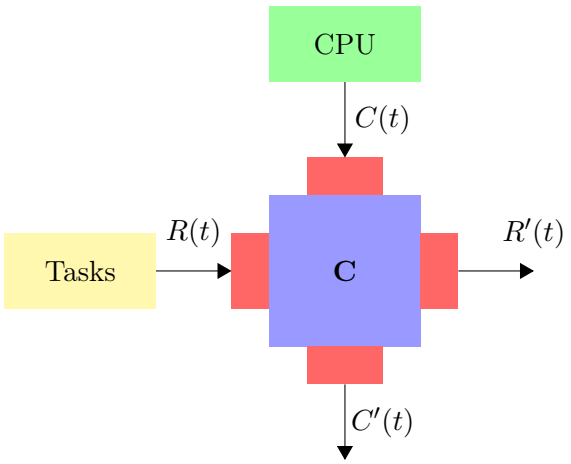

Figure 2.8: Concrete Component with input traces R and C.

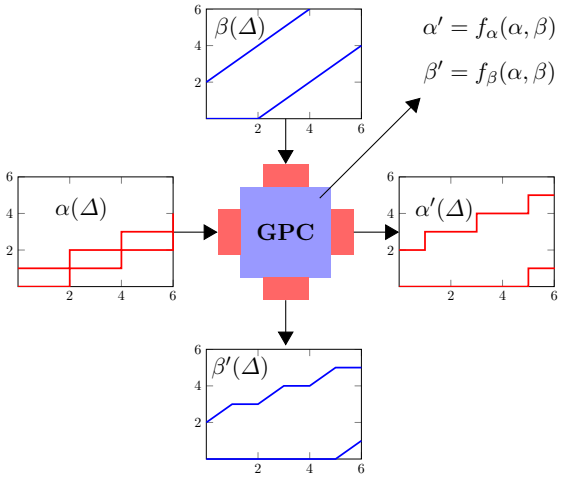

Figure 2.9: Abstract Component processing event and service curve.

we conclude by describing the currently available scheduling policies within the real-time calculus toolbox.

2.2.1 Backlog and Delay

Using the curve representation it is possible to perform an analysis over a model, deriving hard bounds for the system performance. If we consider a GPC component as a single unit, we can calculate the amount of buffer space required b_{max} by the incoming events FIFO queue as well as the maximum delay d_{max} of incoming events. The d_{max} is defined as the maximum amount of time passed from the arrival time of an event to its completed execution. As we can see from figure 2.10, b_{max} represents the maximum vertical distance and d_{max} the maximum horizontal distance between the upper arrival curve α^u and the lower service curve β^l. Formally, the buffer is given by the following equation:

$$b_{max} \leq \sup_{\lambda \geq 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} := \text{Buf}(\alpha^u, \beta^l) \tag{2.12}$$

while the delay is mathematically described by:

$$d_{max} \leq \sup_{\lambda \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\} \right\} := \text{Del}(\alpha^u, \beta^l) \tag{2.13}$$

2.2.2 Network of Components

However, the most important characteristic of real-time calculus is the simplicity of representing a distributed system. From its semantics, it is easy to see that in order to represent a multi-unit system, one has to directly interconnect the abstract components modelling the different parts of the considered architecture.

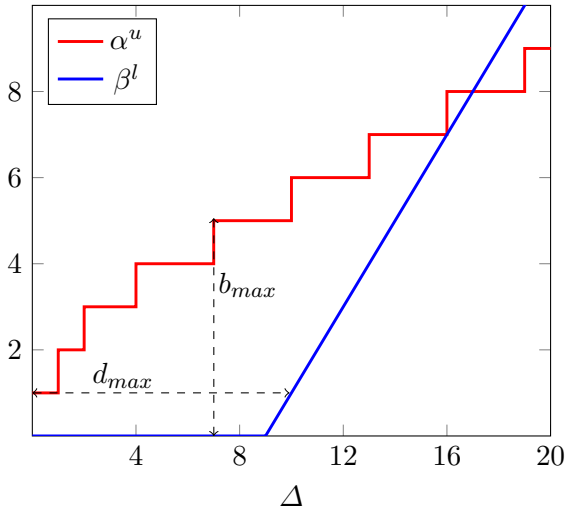

Figure 2.10: Maximum Buffer and Delay

Let us take for example a system consisting of two CPUs sharing a bus, which has to compute a single input event stream. The first CPU receives the events from some external unit, performs a computation for each event and forwards the result to the second CPU, which will also execute the event delivering the final result. Let us assume that the bus implements a TDMA policy, meaning that the communication bus is probably shared with some other resource, about which we do not have any information.

In order to model such a system, we need to firstly derive the outgoing streams. For an incoming event stream $R(t)$ and a resource stream $C(t)$, the definition of the remaining resource stream is straightforwardly given by $C'(t) = C(t) - R(t)$, whilst the outgoing resource stream can be derived by:

$$R(t) = \inf_{0 \leq \lambda \leq t} \{R(\lambda) + C(t) - C(\lambda)\} \tag{2.14}$$

It is clear that if we take the output curves as the input for the next components, we are able to model the system as an interconnection of the components. Moreover, the advantage of using real-time calculus is that it is possible to compute the so called end-to-end delay of the task we are analysing. One option would be to add the d_{max} of every single component, in this case the two CPUs and the bus. However, already in network calculus [7], Le Boudec et al. demonstrate that such a computation for the end-to-end delay is too pessimistic. In fact, the worst-case scenario of an event stream, such as a burst situation, can only appear if the stream is considered as a single entity, whereas a sequence of components cannot: this behaviour is called “pay burst only once”.

2.2.3 Scheduling Policies

The possibility of representing single entities by means of a GPC component does not only represent an advantage for distributed systems, but can also be used in order to model scheduling policies such as fixed-priority.

For example, a FIFO scheduling policy can be achieved by joining together the different events sub-streams, processing them through a GPC, and then using a fork operations to derive the single outgoing sub-streams [28].

Another example of scheduling policy with GPC components is the *pre-emptive fixed-priority* scheduling. Let us assume a set T of τ_i tasks, with $i = 1, ..., n$. The priority is denoted by the subscript with 1 being the highest and n the lowest priority. In order to reproduce this scheduling policy, one needs to construct a top-down chain of GPC components reflecting the priority assignment. The remaining service curve computed for the GPC_i task will be the service curve of the GPC_{i+1} component.

In this way, it is easy to represent any fixed-priority scheduling policy, such as rate monotonic. However, in order to do so, every outgoing service curve has to be derived. Moreover, as we will shown in chapter 3, the complexity of many real-time calculus operations depends on the number of segments needed to represent a curve. It is easy to see that the remaining service curve will be more and more complex to describe, thus exponentially increasing the computation complexity for the lowest priority tasks.

From this observation we based the main idea of this work: instead of considering a fixed-priority scheduling as an interconnection of GPC components, we could perform an analysis of the task set using classical scheduling theory.

Chapter 3

Real-Time Calculus Complexity

As already mentioned in the previous chapters, real-time calculus suffers from an intrinsic drawback. In real-time calculus, the curves have a major role in the performance analysis. In fact, they represent the only input needed in order to model a system: as Stoimenov reports, the curves are "*first class citizens*"[37]. However, many RTC operations are defined for an interval $\Delta \geq 0$, thus obviously leading to a high computational complexity.

Already in Wandeler's Ph.D. thesis [41, p.159], the author points out this problematic:

> The major problem arises from the fact, that the various */curves/* are defined for the infinite range of positive real numbers $\Delta \in \mathbb{R} \geq 0$. However, for practical computation we require that the [curves] have a finite representation, and that applying any of the curve operations leads to a result in a finite time.

3.1 Mathematical Basics

In order to better understand the problem of real-time calculus high complexity, we present in this section its mathematical core, i.e. the min-plus and max-plus algebræ.

3.1.1 Min-Plus and Max-Plus Algebræ

The core of RTC is based on the *min-plus* and *max-plus* algebra. In contrast to the plus-times algebra [41], two operations are defined, *min* (*max*) and *plus*, and replace the traditional addition and multiplication operators.

Formally, if we consider a finite or infinite set of elements $\mathcal{S}$, the min-plus algebra is denoted by $(\mathcal{S} \cup \{\infty\}, \min, +)$. The same logic applies to the max-plus algebra: $(\mathcal{S} \cup \{\infty\}, \max, +)$.

Let $f(t) : \mathcal{S} \longrightarrow \mathbb{R}$ be a function. If we extend the min-plus algebra in the same way as the plus-algebra. The integral operator becomes therefore the infimum (supremum respectively) [7]:

$$\inf_{0\leq s\leq t}\{f(s)\},\ \sup_{0\leq s\leq t}\{f(s)\}$$

Two of the key operations of real-time calculus are convolution and deconvolution. These operations come from the system analysis area and are widely used in the signal processing field. In network and real-time calculus these operations are used in order to relate two input curves f and g. In signal theory, where the domain is continuous, i.e. $t \in (R)$, the integral operator is used to convolute the function, whereas in NC and RTC if we extend the min-plus/max-plus algebra, the convolution is depicted by the infimum/supremum operators, respectively.

3.1.1 Definition (Convolution and Deconvolution). Let f, g be two functions. We define min-plus convolution $\otimes$, min-plus deconvolution $\oslash$, max-plus convolution $\overline{\otimes}$ and max-plus deconvolution $\overline{\oslash}$ as:

$$\otimes : (f \otimes g)(\Delta) = \inf_{0\leq\lambda\leq\Delta}\{f(\Delta-\lambda)+g(\lambda)\} \tag{3.1}$$

$$\oslash : (f \oslash g)(\Delta) = \sup_{\lambda\geq 0}\{f(\Delta+\lambda)-g(\lambda)\} \tag{3.2}$$

$$\overline{\otimes} : (f\overline{\otimes}g)(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{f(\Delta-\lambda)+g(\lambda)\} \tag{3.3}$$

$$\overline{\oslash} : (f\overline{\oslash}g)(\Delta) = \inf_{\lambda\geq 0}\{f(\Delta+\lambda)-g(\lambda)\} \tag{3.4}$$

If we consider for example the min-plus convolution, it is already clear why real-time calculus has a high complexity. In $\otimes$ in order to calculate the convolution of the two functions we need to consider every interval $\Delta \geq 0$ and for each interval we have to shift the variable λ over the interval.

3.1.2 Deriving Incoming Curves

Convolution and deconvolution are used in order to derive the input streams of the model. Without entering into the details, the following equations describe this mathematical relation, for an incoming stream R and a resource C:

$$\alpha^u(\Delta) = R \oslash R \tag{3.5}$$

$$\alpha^l(\Delta) = R\overline{\oslash}R \tag{3.6}$$

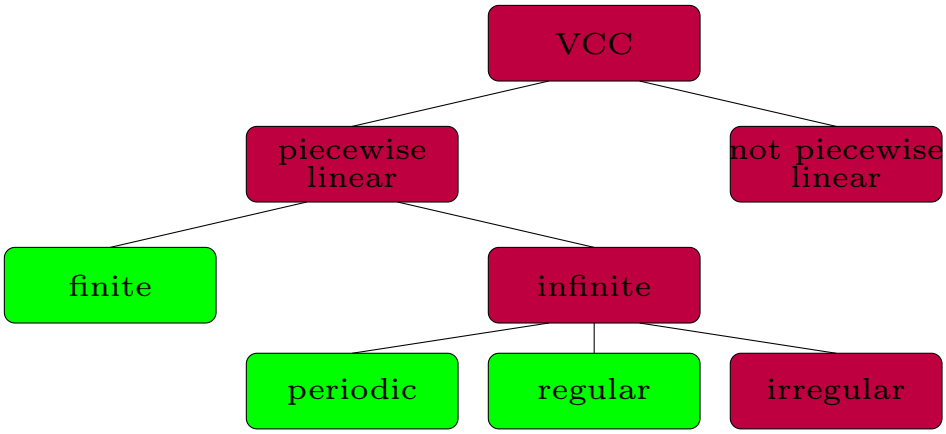

Figure 3.1: Variability characterisation curves taxonomy

$$\beta^u(\Delta) = C \oslash C \tag{3.7}$$

$$\beta^l(\Delta) = C \overline{\oslash} C \tag{3.8}$$

The demonstration is straightforward. Consider $\alpha^u(\Delta)$, from the above relation we have $\alpha^u(\Delta) = \sup_{\lambda \geq 0}\{R(\Delta + \lambda) - R(\lambda)\} \geq R(\Delta + \lambda) - R(\lambda), \forall \Delta \geq 0$, which is its definition from equation 2.1.

3.1.3 Greedy Processing Component

In the same way, real-time calculus describes a greedy processing component with the convolution and deconvolution operations. In order to obtain the output curves the input arrival and service curve are convoluted or de-convoluted. For a proof of these relations refer to Wandeler's thesis [41].

$$\alpha^{u'} = min\left\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\right\} \tag{3.9}$$

$$\alpha^{l'} = min\left\{(\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l\right\} \tag{3.10}$$

$$\beta^{u'} = (\beta^u - \alpha^l) \overline{\oslash} 0 \tag{3.11}$$

$$\beta^{l'} = (\beta^l - \alpha^u) \overline{\otimes} 0 \tag{3.12}$$

3.2 Representation of Curves

It is clear that defining curves for a $\Delta \geq 0$ yields a problem in terms of the curve representation. The incoming streams need to be somehow represented, i.e. it is not

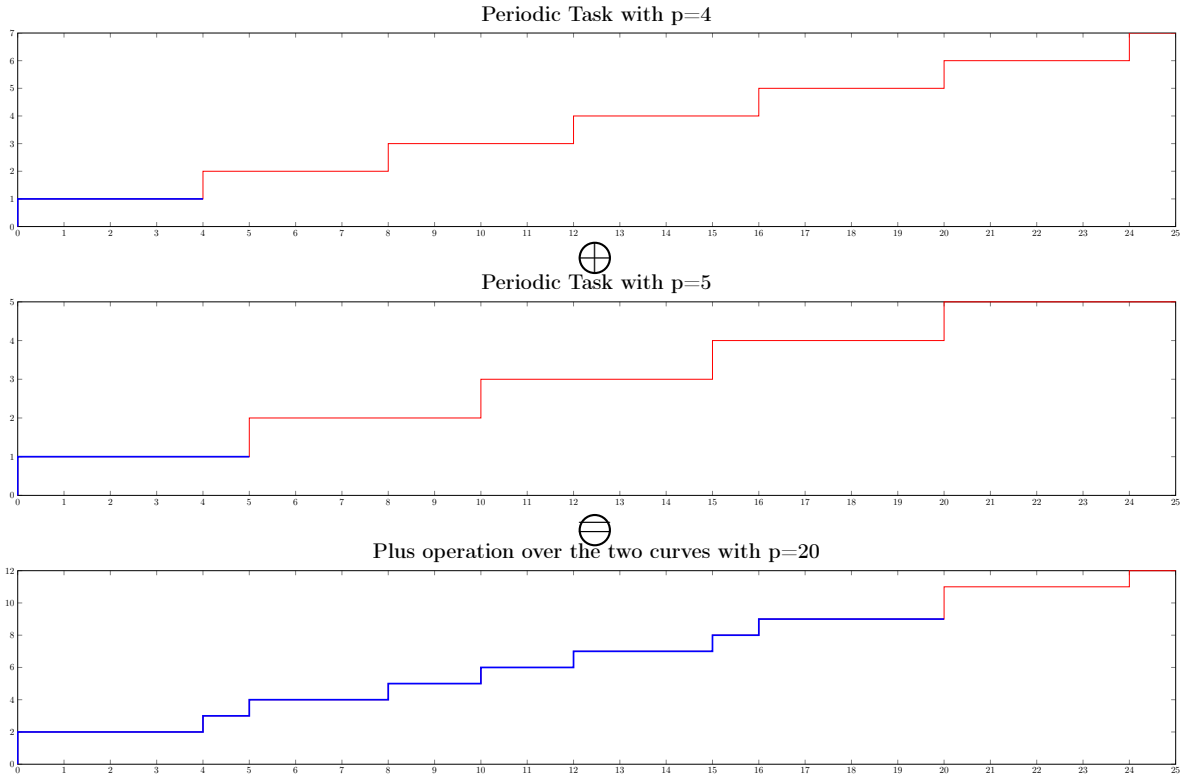

Figure 3.2: Period Explosion Phenomenon

possible - since they need a finite data structure - to work with infinite curves. In fact, the original Matlab Toolbox for real-time calculus already addresses this problem and suggests a solution for it [40].

The RTC-Toolbox implements a data structure in order to represent only a subset of the so called variability characterisation curves (VCC). For a given increasing function, such as the concrete stream R, the VCCs are defined as the upper and lower curves bounding R, like α^l and α^u. Wandeler in [41] defines a taxonomy over the set of the VCCs.

In the area of piecewise linear functions, the finite piecewise linear curves are those depicted by a single linear function, like for example the curve for a fully available resource. The irregular curves of the infinite class cannot be expressed mathematically for obvious reasons. Periodic curves such as a those for periodic tasks or for the TDMA policy have a compact representation since it is possible to just "repeat" the period over and over. More interesting are the regular curves. In this case the curves are divided into two parts: an aperiodic part, for example for the burst region, and a periodic part which reflects the long term behaviour of the curve.

The complexity of real-time calculus depends precisely on this representation: the computation time of many RTC operations is proportional to the number of segments needed to represent a curve [17]. Moreover, the complexity particularly increases if the

two considered curves have a period prime to each other[41]. Guan et al. in [17] define this problem as a *period explosion*.

For example, let us take a simple plus operation over two periodic curves, with a period of 4 and 5 time units respectively: the resulting curve will already have a period of 20 time units, and the number of segments needed passes from one for each curve to a total of 8 segments (See figure 3.2). In fact, even in the current RTC-Toolbox implementation, if the period increases with every operation, the memory needed for the curves data structure will quickly reach the maximum amount available, yielding the `OUT OF MEMORY`-error in the Java-Kernel of the framework [17].

3.3 Finitary Real-Time Calculus

In 2013 Guan et al. [17] proposed an extension to real-time calculus named finitary real-time calculus. The idea behind this extension, is that it is not always needed to work on curves defined for a $\Delta \geq 0$, but it is possible to achieve the same results as in the original framework by only considering the curves up to a point called *maximal busy window slice* (MBS). Its visual representation is depicted in figure 3.3.

This concept does not only work when calculating the maximum delay d_{max} and the maximum buffer b_{max}, but it is also extended for GPC components. In this case however, the finitary real-time calculus analysis becomes more complex. In fact, if we consider a single GPC component, the analysis for an interval Δ is bounded by the interval $[\Delta, \Delta + MBS]$ and $[0, MBS]$ depending on the operation and curves considered, thus reducing the framework computation demand.

Nevertheless, a problem arises when considering a network of components. Finitary real-time calculus defines a so called finitary deconvolution which bounds *only* the λ variable. Yet, it also needs to bound the Δ variable, i.e. cutting off the curve after some point. In this circumstance, it is not possible to bound the Δ a priori, as we need to ensure that the outgoing curves will be defined at least for the MBS of the next component.

The authors define a theorem, which states that in order to describe an output curve between $[0, x]$, we need to consider the incoming α and β over a range $[0, x + T]$, where $T \geq MBS(\alpha, \beta)$ (see theorem 4 in [17]). This means, if we know the maximum busy window slice of the last component of the network, we could proceed backwards in order to find the MBS of each component of the model. However, this is impractical, since knowing the MBS of the last component means analysing the whole network as a normal RTC-model. In fact, we would need to derive every outgoing stream of every component.

To solve this problem, the authors of finitary real-time calculus propose a three-steps strategy for analysing a network of components:

1. use a linear approximation of the incoming stream in order to quickly analyse the network and compute an MBS upper bound for every component;

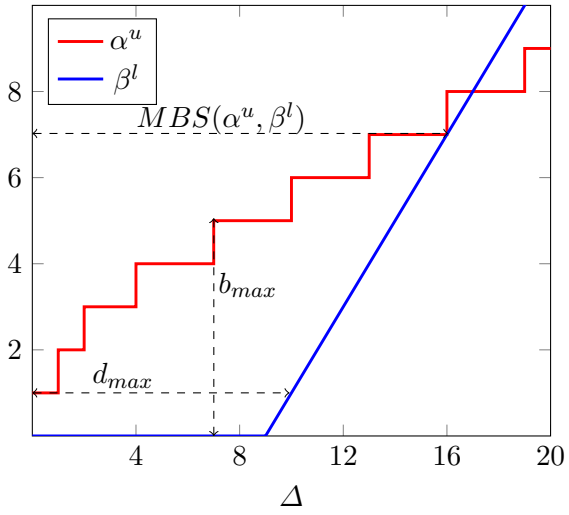

Figure 3.3: Maximum Busy Window Slice

2. proceed backwards to find the "truncation" point of every curve;

3. analyse the model with finitary deconvolution and convolution, i.e. with finitary GPC;

The complexity of this procedure is pseudo-polynomial. Practically, finitary real-time calculus drastically increases the computation time needed to analyse a network. In particular, the evaluation shows that finitary real-time calculus does not depend on the period of the considered curves, whereas the time needed by the original RTC framework rapidly grows depending on the hyper-period of the incoming curves.

However, as the authors underline, a higher utilisation of the considered curve set needs a much higher computation time. If the total utilisation stays under 80%, finitary RTC speeds the computation up to 1000 times, while after this threshold it drops to factor 20. Such a behaviour is explained by the fact that in case of higher utilisation, the remaining service curves will have much longer periods, thus shifting the MBS point to much bigger values.

3.4 Curve Approximation

Finitary real-time calculus improves drastically the original real-time calculus in terms of efficiency. As the evaluation in [17] shows, the practical benefits decreases with the growth of the component utilisation. Other methods use the approximation of the curves in order to reduce the RTC-Complexity, however this is done at the detriment of the exactness.

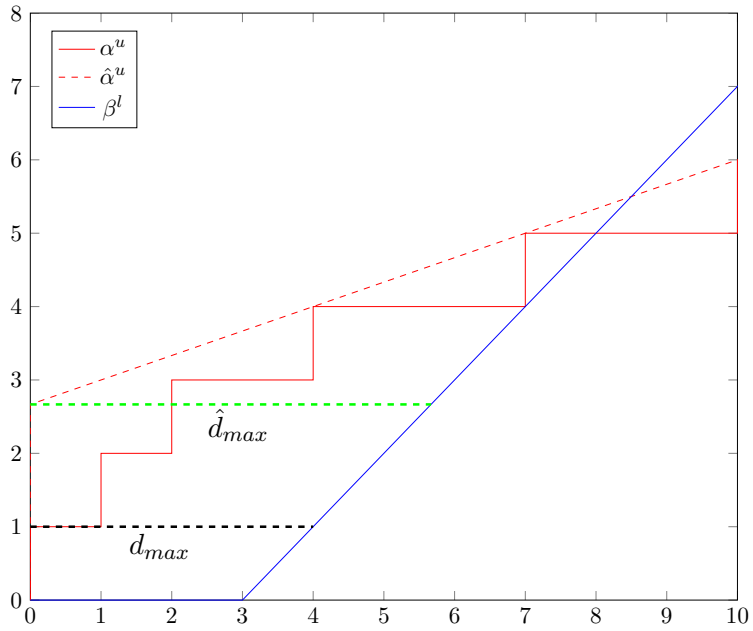

Figure 3.4: Delay comparison between the original arrival curve and its approximation

3.4.1 Linear Approximation

The easiest way to approximate an arrival curve is done by means of its slope. To derive an approximation of the curves, the idea is to first obtain the slope of the curve and then to calculate the smallest y-interception of the line equation, so that the tightest over-approximation line bounding the original curve can be defined.

The slope of a function is defined as:

$$s(f) \triangleq \lim_{\Delta \to \infty} \left(\frac{f(\Delta)}{\Delta} \right) \tag{3.13}$$

The approximation can then be obtained as follows. Let $y = mx + b$ denote the equation of the line. Given any event stream function $f(\Delta)$, we set:

$$m = s(f(\Delta)) \tag{3.14}$$

$$b = inf\{b^u \mid \forall \Delta : mx + b^u \geq f(\Delta)\} \tag{3.15}$$

This approach has been used by Guan et al. in [17] in order to perform a pre-analysis over a network of components. The complexity for finding the tightest over-approximation of the arrival curve is pseudo-polynomial.

Even though the computation complexity of the framework remains high, this simple method allows a quick analysis of a system model. In addition, since real-time calculus already adds some pessimism to the system performance analysis, this method appears to be unsuitable for performing an end-to-end delay computation of the system [2].

Moreover, Wandeler in his Ph.D. thesis points out how pessimistic this approximation can be. If we consider a simple example as in figure 3.4, where the arrival curve is represented by a pjd curve with $p = 3$, $j = 5$ and $d = 1$ and the service curve models a bounded delay with $D = 3$ and $B = 1$, the corresponding delay is originally calculated for $d_{max} = 4$, whereas its value for the approximated α^u is $\hat{d}_{max} = 5,667$, which means it is already overestimated by $\sim 41\%$ more than the exact delay.

For this reasons, a simple linear approximation of the arrival curve appears to be insufficient for a satisfactory analysis. As we have seen, the complexity of real-time calculus depends on the number of segments describing the curve, therefore a linear approximation drastically decreases it, since we now have only one segment depicting the curve. Nevertheless, already in simple cases the approximation error is too pessimistic. Moreover, with this method we are not able to bound the over-estimation given by the approximate analysis. Therefore, we do not consider linear approximation as a suitable method for the aims of this work.

Chapter 4

Scheduling Theory

As Guan et al. shows in [17], finitary real-time calculus reduces the complexity of the overall RTC-Framework from exponential to pseudo-polynomial. This has been done by considering only the so-called *maximum busy-period size* (MBS): Guan proves in the paper [17] that the maximum delay and the maximum backlog always take place within this initial region. This means that *cutting off* the curve after a determined point does not influence the overall analysis.

Nevertheless, since the MBS has to be calculated for each and every component of the system, the outgoing streams need to be defined over an interval long enough to contain the MBS of the next one. For some operations however, the number of components needed in order to analyse the system can be reduced using classical scheduling theory, e.g. in order to get a simple feasibility analysis over a task set or to derive the worst-case response time of a particular task: this represents the main contribution of this work.

In fact, if we consider a fixed-priority component (FP) in real-time calculus, we notice that it is built by interconnecting two or more greedy processing components, with the top-down order denoting the priority level of the tasks. This means, that in order to analyse the delay and buffer from the second highest priority task onwards, we have to obtain every remaining service curve of the higher priority tasks. In this manner, even a schedulability test over a task set is of exponential complexity (see also figures 1.1 and 1.2).

In the field of scheduling theory, such task models are widely researched and constantly improved. For this reason, we turned our attention to these results in order to find a possible integration of scheduling theory with the real-time calculus framework.

4.1 Classical Scheduling Theory

Traditional scheduling theory does not consider curves and streams. Instead, a schedulability problem Q is defined mainly over a given task set of n tasks $T = \{\tau_1, \tau_2, ..., \tau_n\}$, a set

of m processors $P = \{P_1, P_2, ..., P_m\}$ and a set of s types resources $R = \{R_1, R_2, ..., R_s\}$. A schedulability analysis over the problem Q aims to assign for every task τ_i of T a processor P_i and a resource R_i, such that all the tasks can be completed within certain constrains [8].

The task constraints are determined by different task models and system assumptions. The following list gives an overview of the most common assumptions:

Static vs Dynamic: in a static system scheduling decisions are taken based on fixed parameters, that do no change during the execution of the system. Instead, a dynamic system bases its scheduling decisions on parameters that can change over time;

Pre-emptive vs Non-Pre-emptive: a pre-emptive task means that its execution can be interrupted at any time by another task, according to a determined policy, even if it did not finish its computation. A non-pre-emptive task will instead continue its execution until the end;

Optimal vs Heuristic: an optimal algorithm tries to find the best task schedule for a given parameter. Instead a heuristic algorithm aims to return any feasible schedule, but it does not guarantee that this would be the best solution.

In this work we are mainly interested in static-priority pre-emptive scheduling algorithms. This set is composed by the widely known and researched fixed-priority scheduling policies. In the next section we formally define the task model that we analysed for this thesis.

4.2 Task Model

Before explaining the properties of rate monotonic and deadline monotonic, we shall introduce the notation and the semantic of the considered task model. A set T of n tasks is defined by $\tau_i : i = 1, ..., n$ tasks. Each task can release more than one job, with $\tau_{i,j}$ we denote the jth instance of task τ_i. The time point at which the jth job of task τ_i becomes available is called arrival time $a_{i,j}$. The term is also reported as release time $r_{i,j}$.

The amount of time needed by the processor to fully complete each job of τ_i is given by e_i. The time at which the job has to be completed w.r.t. its release time r_i, is called relative deadline D_i. This means, that the system has to complete the job before this time. Otherwise, the task is said to not be schedulable and can lead to the consequences reported in chapter 1. The finishing time of a job is given by $f_{i,j}$.

Another important metric parameter is the response time of a task. The response time is defined by the finishing time and the release time, i.e. $R_{i,j} = f_{i,j} - r_{i,j}$. The worst-case response time of a task τ_i is intuitively defined by $R_i = \max_{j>0}(R_{i,j})$.

In the following sections we mainly consider periodic tasks, i.e. every task τ_i is composed by an infinite sequence of jobs, which are regularly released. We define p_i as the period of task τ_i. Analogously, the release time of a job could undergo a so called jitter (see also chapter 2). This means that the arrival time can be shifted at most by a $\pm j_i$. Usually the minimum inter-arrival time of a periodic task is given by its period. Nevertheless, in cases where a jitter is defined, a parameter $d_i < p_i$ can be used to describe a minimum inter-arrival time between tasks that are subject to a long jitter, i.e. $j_i > p_i$.

To summarise, a periodic task can be completely described by the tuple (p, j, d), which can be used in order to derive a worst-case response time R_i, w.r.t. a task set T. The R_i can then be compared to the relative deadline D_i of each task in order to verify if the task set is schedulable. To be more precise, there are three types of relative deadlines which are considered in the literature:

implicit deadline: the relative deadline is equal to the period of the task, i.e. $D_i = p_i, \forall \tau_i \in T$;

constrained or bounded deadline: the relative deadline is equal to, or smaller than the period of the task, i.e. $D_i \leq p_i, \forall \tau_i \in T$;

arbitrary deadline: the relative deadline is no longer bounded to the period, i.e. D_i can also assume larger values than $p_i, \forall \tau_i \in T$.

Furthermore in order to derive R_i, we also need to consider the scheduling policy of T. As mentioned above, a scheduling policy returns an assignment of which tasks has to run on a processor at a certain point in time. A fixed-priority pre-emptive scheduling policy for periodic tasks with or without jitter or inter-arrival time returns a schedule depending on the priority level π_i of the task τ_i. Let us assume a priority ordering over the $\tau_i : i = 1, ..., n$, i.e. for each $k < j$, then task τ_k has a higher priority than τ_j. The set $hp(\tau_i)$ denotes all the tasks that have higher priority than τ_i. According to a fixed-priority pre-emptive scheduling, a higher priority task will always interrupt at its release time the execution of a lower priority task. That means, a task τ_i can be executed if no job of the tasks $hp(\tau_i)$ is currently waiting to be executed.

4.2.1 Rate Monotonic Scheduling Algorithm

The problem now is shifted to how to determine the priorities. The rate-monotonic algorithm assigned to each task a priority depending on the period of the tasks. For periodic tasks with implicit deadlines, the smaller the period, the bigger the priority assignments. The RM algorithm has been proven by Liu and Leyland [22] to be optimal among the class of static priority tasks with implicit deadlines, i.e. $D_i = p_i$. This means, that if a fixed-priority algorithm can schedule a given task set, than this set is also schedulable

by RM. Moreover, the authors derive in the same paper the least upper bound for the processor utilisation over a task set. The utilisation factor of a task τ_i is defined by:

$$U_i = \frac{e_i}{p_i} \tag{4.1}$$

and subsequently the total utilisation of a task set is given by the sum of all tasks' utilisation:

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} \tag{4.2}$$

The least upper bound U_{lub} defines the minimum utilisation factor over all sets of tasks that fully utilise the processor, i.e. below this bound every task set is schedulable by RM, whereas above this bound a task set is schedulable only for suitable periods of the tasks. Formally, for RM the U_{lub} is described for a set of n tasks by:

$$U_{lub} = n(2^{\frac{1}{n}} - 1) \tag{4.3}$$

which for $n \longrightarrow \infty$ converges to $\simeq 0,69$.

The least upper bound gives a quick schedulability test for a task set. However, this test is not exact. In fact, a task set can also be schedulable if its total utilisation exceeds the U_{lub}. This represents the so called *utilisation-based schedulability test*, which is sufficient, but not an exact test for the rate monotonic. For this reason, a *workload analysis* has been developed by Lehoczky et al. in [21]. These methods are based on the analysis of the workload function between a requested interval, typically between $(0, D_i]$, that is from the release time of the task and its deadline. If between these interval there exists a point where the workload function lies below the time function $f(t) = t$ denoting a 100% available processor, then the task set is schedulable by a fixed-priority algorithm. We do not go into details in this section, since the workload function will be the core of the approximate algorithm for the PJD* component. It is however important to underline that the schedulability test for the RM algorithm based on the workload analysis is of pseudo-polynomial complexity.

4.2.2 Deadline Monotonic Scheduling Algorithm

Another widely used priority assignment technique is the so called *deadline-monotonic* algorithm. In this case the priorities of each tasks are assigned w.r.t. their relative deadline, i.e. the shorter the deadline, the higher the priority is. For such task sets the DM algorithm is proven to be optimal [21].

For the DM algorithm the workload analysis is even more important; in fact the total utilisation for a task set is now bounded by:

$$U = \sum_{i=1}^{n} \frac{e_i}{D_i} \leq n(2^{\frac{1}{n}} - 1) \tag{4.4}$$

Task Model	Complexity
implicit deadlines	pseudo-polynomial
constrained deadlines	pseudo-polynomial
arbitrary deadlines	exponential

Table 4.1: Complexity for sporadic task models [16].

which has however been shown to be too pessimistic [8].

To conclude this overview of priority assignments, we consider a set of tasks subject to arbitrary deadlines. In this task model we can no longer assume that the worst-case scenario of the task set happens when all of the tasks release a job at the same time. In fact, for the implicit and constrained deadlines model, the analysis is based on the fact that the worst-case response time of a task τ_i happens when all the other higher priority tasks simultaneously release a job. Then, we only need to consider the first job release by task τ_i. This represents the so called *critical instant theorem*.

Instead, for an arbitrary deadlines model, this is not true. In fact, as Lehoczky shows in [20], the worst-case response time can also take place for a later job release of task τ_i. In this situation we have to take into account the number of job releases in the *level-i* busy period, which represents the period within which only jobs of tasks of priority greater or equal than i are being executed. We will return on this particular task model in chapter 5.

4.3 Polynomial-Time Approximation Scheme

As we have already mentioned above, the feasibility test for static-priority and sporadic task models is known to be at best of pseudo-polynomial complexity [16]. Pseudo-polynomial means that the test runs in polynomial time if the input is given in the numeric form - i.e. base 10 -, but it is exponential in the binary form, that means, it depends on the number of bits necessary to represent the given input.

For sporadic systems with implicit deadline Lehoczky et al. presented in [21] a pseudo-polynomial feasibility test. The same complexity is reached by the feasibility test of Audsley et al. in [4] for sporadic tasks with constrained deadlines. The overall high complexity is due to the fact that the test is in fact of polynomial complexity, since it depends on the values of the task, which means that the test is of pseudo-polynomial complexity, since it depends on the length of the input.

Moreover, for task systems with arbitrary deadlines Lehoczky shows [20] that the complexity for the feasibility test is exponential. Table 4.1 reports the complexity for the various cases.

In order to reduce the computational complexity of such algorithms, it is sometimes possible to trade the *exactness* of the computation with the running time of the decision test. The idea is to design an algorithm which runs in polynomial time and *approximates* the optimal algorithm by a *ratio* $\rho(n)$: that is called a $\rho(n)$-*approximation algorithm* [11].

The ratio $\rho(n)$ is given by considering the computation cost C and $C*$ of the optimal and the approximation algorithm respectively. The maximum function considers both maximisation and minimisation problems, i.e.:

$$\max\left(\frac{C}{C*}, \frac{C*}{C}\right) \leq \rho(n) \tag{4.5}$$

In this context an *approximation scheme* denotes an approximation algorithm that takes a problem Q and a value $\epsilon > 0$ that defines the approximation ratio of the algorithm, i.e. for a fixed ϵ we will obtain a $(1-\epsilon)$-approximation algorithm.

If the scheme runs in polynomial time for every $\epsilon > 0$ with respect to the size of the input n, the approximation scheme is said to be a *polynomial-time approximation scheme* (PTAS). Furthermore, since with the decrease of the factor ϵ the algorithm may change its running time very rapidly, i.e. consider a $O(n^{(2/\epsilon)})$ algorithm, we also want to bound the running time by a constant factor which decreases/increases as ϵ decreases/increases.

If a polynomial-time approximation scheme runs in polynomial time both over the size n of the input and the factor $1/\epsilon$, than it is referred to as a fully polynomial-time approximation scheme.

4.4 FPTAS for Static-Priority Tasks

The fully polynomial-time approximation scheme has been applied to the feasibility problem of static priority tasks. The goal is to somehow bound the total number of points that needs to be checked in order to prove the schedulability, as it originally depends on the task period [15].

Albers et al. in [1] derive a fully polynomial-time approximation scheme for sporadic tasks in order to solve the feasibility problem for dynamic scheduling algorithms. Following this approach, Fisher et al. in [15] extend this work to the domain of fixed priority algorithms.

The idea is to approximate the request-bound function by means of a given acceptance error ϵ, which also gives the approximation level of the request-bound function. Intuitively, the fully polynomial-time approximation scheme represents an algorithm to compute the feasibility of a system for constrained deadline synchronous periodic tasks. The complexity of the algorithm is not based on the number of periods any more, but instead it depends on the number of tasks n and the number of steps k, for which the request-bound function is exactly represented. Formally, the complexity for checking the feasibility of a system is $O(n^2k)$.

In the next chapter, we present an FPTAS, which covers most of the task models implemented by real-time calculus. In particular, we extend the current results to a periodic task with burst model, so we can analyse a set of tasks in a new way than the FP component implemented by the RTC-Toolbox.

Chapter 5

Real-Time Calculus and Classical Scheduling Theory

Even though real-time calculus offers a powerful framework to model distributed systems, the complexity of some of its operation can be reduced in order to perform an analysis over a single component. In this chapter we present a novel approach to implement a fixed-priority scheduling policy for a single component.

5.1 A new Component: PJD*

The computation of every remaining service curve is not strictly necessary for performing certain operations over a task set. For example, let us suppose that our FP component aims to model a Rate Monotonic or Deadline Monotonic scheduling policy. Assuming we have three tasks, in order to compute the delay for the third task, we need to derive two additional remaining service curves, one after the other.

Nevertheless, if we take a look at classical scheduling theory, the fixed-priority scheduling policy has been extensively researched over the last 40 years. From Liu and Layland's bound in [22] to Buttazzo's textbook [8], this area of study has prospered and the available literature, use cases, and new developments are countless.

The idea of combining the available results in real-time calculus operations presented in this thesis stems from this background. In particular, the widely known rate monotonic (RM) and deadline monotonic (DM) fixed-priority scheduling policies have been investigated in order to derive a new component for real-time calculus.

The PJD* component is based on the widely used GPC component of real-time calculus. We will consider an incoming event curve described by the parameters p, j, and d, and a resource curve modelling a fully available processor or a bounded delay behaviour. We will show how to introduce classical schedulability analysis in order to perform a

pseudo-polynomial approximate feasibility test and to derive an approximate worst-case response time.

The development of this work will start from the basic task model, such as the periodic one, up to the more general periodic tasks with burst. We will also analyse both a fully available processor and the bounded delay model.

5.2 FPTAS for Periodic Tasks

As already presented in chapter 2, real-time calculus mathematically describes certain task models in order to easily represent some common task patterns. We could therefore apply the extensive results in the field of classical scheduling analysis to these particular task models. However, while strictly periodic tasks are widely studied, be it for implicit, bounded, or arbitrary deadlines, periodic tasks with jitter are not so commonly found in the literature. Moreover, the periodic task with burst is, to the best of our knowledge, only treated within the context of formal methods and compositional analysis, such as in [32], [19], [28] and [29].

Therefore, in order to cover all the RTC event models, we need to extend the current results of Fisher et al. in [15] and [14], and Nguyen in [25], [27] and [26] to the domain of periodic tasks with jitters and in particular to periodic tasks with burst.

The easiest case to analyse is a task set T of strictly periodic tasks and constrained deadlines. Classical scheduling theory uses a so called *request-bound function* (RBF) to bound the total execution request of a task τ_i at time t:

$$RBF(\tau_i, t) := \left\lceil \frac{t}{p_i} \right\rceil e_i \tag{5.1}$$

The cumulative request-bound function represents the sum of all execution requests of the set $hp(\tau_i)$ denoting all task with a higher priority than τ_i. The equation defining this function is given by:

$$W_i(t) := e_i + \sum_{\tau_j \in hp(\tau_i)} RBF(\tau_j, t) \tag{5.2}$$

The exact schedulability test for a periodic task set with constrained deadlines using a fixed-priority algorithm is reported by Buttazzo in [8], which reformulates the original theorem of Lehoczky, Sha, and Ding in [21] and states:

Theorem (1). *A set of fully pre-emptive tasks with constrained deadlines is schedulable by a fixed-priority algorithm if and only if*

$$\forall i = 1, ..., n, \exists t \in (0, D_i] : W_i(t) \leq t \tag{5.3}$$

From this background, Fisher and Baruah firstly derived in [15] FPTAS for a set of periodic tasks with bounded deadlines. The idea is based on the observation that the

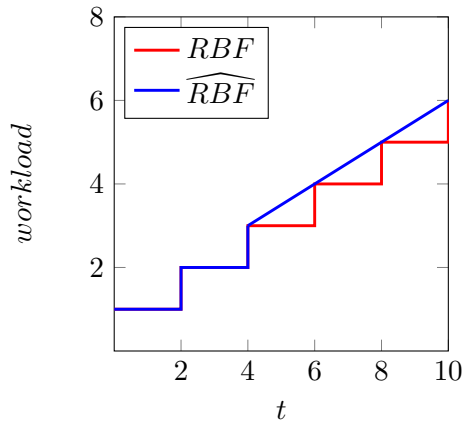

Figure 5.1: Approximated RBF for a $k = 3$ factor

$RBF(\tau_i, t)$ is discontinuous every p_i time units. After a certain numbers of steps the RBF is approximated by simply relaxing the ceiling function in order to get a linear approximation. The number of steps, for which the RBF is being represented exactly, depends on the approximation factor $0 < \epsilon < 1$ and is formally defined by:

$$k := \left\lceil \frac{1}{\epsilon} \right\rceil - 1 \tag{5.4}$$

Intuitively, the FPTAS takes ϵ as input along with the task set that has to be analysed. This parameter will define the $k-1$ steps where the RBF is exactly computed. The function is mathematically described by $\widehat{RBF}$ as follows:

$$\widehat{RBF}(\tau_i, t) = \begin{cases} RBF(\tau_i, t) & \text{if } t \leq (k-1)p_i \\ (t + p_i)\frac{e_i}{p_i} & \text{otherwise} \end{cases} \tag{5.5}$$

Let us suppose an approximation factor ϵ which gives a $k = 3$, then figure 5.1 gives a visual representation of the exact and approximated functions.

Consequently, the cumulative request-bound function has to be modified as well in order to consider the now approximated RBF. This is done by simply substituting in equation 5.2 the RBF with the $\widehat{RBF}$ function:

$$\widehat{W}_i(t) := e_i + \sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}(\tau_j, t) \tag{5.6}$$

At this point, if we use the approximate cumulative request-bound function along with theorem 1, we will no longer obtain an exact feasibility test for our task set. In fact, the $\widehat{W}_i(t)$, can now exceed the original $W_i(t)$ in terms of the intersection point with $f(t) = t$. Moreover, we cannot guarantee the infeasibility of the task set any more. It could be possible that an intersection point is not found between $t \in (0, D_i]$, but the task can still be scheduled.

For this reason, two other theorems are used for the analysis together with the approximate cumulative request-bound function:

Theorem (2). *In a synchronous periodic task set, task τ_i is feasible under a deadline monotonic scheduling policy if:*

$$\exists t \in (0, D_i] : \widehat{W}_i(t) \leq t \tag{5.7}$$

With the FPTAS, we cannot guarantee the infeasibility of the task. Nevertheless we can bound the infeasibility to a slower processor, that means:

Theorem (3). *In a synchronous periodic task set, task τ_i is infeasible under a deadline monotonic scheduling policy for a processor with $(1-\epsilon)$ of the original capacity, if:*

$$\forall t \in (0, D_i] : \widehat{W}_i(t) > t \tag{5.8}$$

To be precise, Fisher states in [15, p.7] that "*we must effectively ignore $(1-\epsilon)$ of the processor capacity for the test to become exact* ".

In order to prove these theorems, it is more practical to first prove a set of properties and lemmas, so that the theorems will easily follow. We report here only their statements, as they will be similarly extended to comprehend different tasks models in the following sections. For the original proof refer to [15].

Property (1). *$\forall t \geq 0$, $\widehat{RBF}(\tau_i, t) \geq RBF(\tau_i, t)$. That means, the approximate request-bound function is an upper bound for the exact request-bound function.*

Property (2). *If $\widehat{RBF}(\tau_i, t) > RBF(\tau_i, t)$, then we have calculated at least $k-1$ steps of the arrival curve, i.e. $RBF(\tau_i, t) \geq ke_i$*

Property (3). *$\forall t$, then the approximation never exceeds the arrival curve by more than one step size, i.e. $\widehat{RBF}(\tau_i, t) - RBF(\tau_i, t) \leq e_i$.*

Property (4). *The last property shows that the approximated function is bounded above by the request-bound function multiplied by the factor $\frac{k+1}{k}$, that is $\forall t \geq 0$, $RBF(\tau_i, t) \leq \widehat{RBF}(\tau_i, t) \leq \left(\frac{k+1}{k}\right) RBF(\tau_i, t)$.*

Lemma (1). *If $\widehat{W}_i(t) \leq t$, then $W_i(t) \leq t$. That is, if the approximate cumulative request-bound function is below $f(t) = t$, therefore the exact cumulative request-bound function is below $f(t) = t$ as well.*

Lemma (2). *If $\widehat{W}_i(t) > t$, then $W_i(t) > \frac{k}{k+1}t$, i.e. if the approximate cumulative request-bound function is above $f(t) = t$, thus the exact cumulative request-bound function is above the line $f(t) = \frac{k}{k+1}(t)$.*

5.2.1 Testing Set for Periodic Tasks

While approximating the request bound function simplifies the analysis over the long term curve, the complexity of the feasibility test still remains high, i.e. pseudo-polynomial. This can be overcome by limiting the testing set for which the functions $W_i(t)$ and $\widehat{W}_i(t)$ have to be evaluated. In fact, since the cumulative request-bound function is a step function, we would need to evaluate it intuitively only at its edges as reported in [21] and [8].

However, the test still runs in pseudo-polynomial time, since the number of testing points depends on the period of the tasks. The exact testing set for a task τ_i in a periodic task model is defined by Fisher and Baruah in [15] as:

$$\mathcal{S}_i := \left\{ t = bp_a : a = 1, ..., i; b = 1, ..., \left\lfloor \frac{D_i}{p_a} \right\rfloor \right\} \tag{5.9}$$

The key point of the FPTAS lies precisely in this set. In fact, given the approximation rate ϵ we can bound the testing points to the parameter k. Therefore, the testing set is not dependent any more on the task periods, but on the number i of tasks and the number k of steps for which the request-bound functions are exactly computed.

Formally, given an approximation factor ϵ, the approximate cumulative function $\widehat{W}_i(t)$ needs to be tested for the points defined in the following set [16]:

$$\widehat{\mathcal{S}}_i := \{t = bp_a : a = 1, ..., i-1; b = 1, ..., k-1\} \cup \{t = D_i\} \tag{5.10}$$

For the approximate testing set, we need to additionally prove that the set is sufficient. That means, if $\forall t \in \widehat{\mathcal{S}}_i, \widehat{W}_i(t) > t$, then $\forall t \in (0, D_i], \widehat{W}_i(t) > t$. Let us assume a pair of testing points $t_1, t_2 \in \widehat{\mathcal{S}}_i$, with $t_1 < t_2$. We then call this points *adjacent*, if no other scheduling point in $\widehat{\mathcal{S}}_i$ lies between them, i.e. there exists no point $t \in \widehat{\mathcal{S}}_i$ such as $t_1 < t < t_2$. If for these two adjacent points we have that t_1, t_2 in $\widehat{\mathcal{S}}_i$, for which $\widehat{W}_i(t) > t_1$ and $\widehat{W}_i(t) > t_2$, then, since the cumulative function in non decreasing, we have that for every time point t in the interval $(t_1, t_2), \widehat{W}_i(t) > t$. This result can easily be extended to all the points in $\widehat{\mathcal{S}}_i$, thus proving the statement.

5.2.2 Response Time Analysis for Periodic Tasks

To conclude the analysis for the periodic task model, we present the response time analysis applying to it. Since the schedulability analysis for theorem 1 is in fact based on a time-demand analysis, worst-case response time can be derived using the cumulative request-bound function and the resource function $f(t) = t$.

In [31], Richard et al. extended the feasibility analysis of Lehoczky, Sha, and Ding [21], in order to derive the exact worst-case response time of a feasible task.

The points derived by the testing sets are defined as *scheduling points* of the task set. In particular, we are interested in the *critical scheduling point*, which is defined as:

5.2.5 Definition (Critical Scheduling Point). For a task τ_i, the critical scheduling point over its testing set $\mathcal{S}_i$ is defined as:

$$t^* := \min\{t \in \mathcal{S}_i : W_i(t) \leq t\} \tag{5.11}$$

If such a point exists, we can thus state the relationship between point t^* and the wcrt as follows:

5.2.6 Theorem (wcrt). *For a feasible task τ_i, the worst-case response time is determined by its critical scheduling point t^*.*

Proof. *If the task is feasible, then the point t^* exists within the testing set $\mathcal{S}_i$. If we depict the testing set as $\mathcal{S}_i = \{t_{i1}, ..., t_{il}\}$ in an increasing order and with t_{il} denoting the deadline D_i of the task, the critical scheduling point would be $t^* = t_{ij}$, with $1 \leq j \leq l$. This point is the first scheduling point such as $W_i(t) \leq t$, so that $\forall t \in \{t_{i1}, ..., t_{ij-1}\}$, then $W_i(t) > t$ and $\forall t \in \{t_{ij}, ..., t_{il}\}$, therefore $W_i(t) \leq t$. Since the cumulative request-bound function $W_i(t)$ is non decreasing, there exists a point t in the interval $(t_{ij-1}, t_{ij}]$ such that $W_i(t) = t$. The scheduling points represent the task release, therefore there would not be any task release between t and t^*, thus $W_i(t) = W_i(t^*)$. The worst-case response time is thus defined by $R_i = W_i(t^*)$.*

As the authors point out, this method only works for feasible tasks. In fact, if the feasibility test fails, we would have no point in $\mathcal{S}_i$ intersecting the function $f(t)$, hence we would have to consider more testing points for the task.

In the approximate analysis we are more interested in bounding the approximate worst-case response time for a given approximation factor ϵ, in the same way as in the approximate cumulative function analysis. First, we define the approximate worst-case response time.

5.2.8 Definition (Approximate worst-case response time). For a task τ_i, the approximate worst-case response time is defined as:

$$\widehat{R}_i := \min\{t > 0 : \widehat{W}_i(t) = t\} \tag{5.12}$$

The proof that $\widehat{R}_i$ is bounded follows from the two following lemmas, which are defined by Nguyen et al. in [26]:

1 Lemma. *Let us assume a task τ_i with relative deadline D_i, then:*

$$\widehat{R}_i \leq D_i \iff \exists t \in \widehat{\mathcal{S}}_i : \widehat{W}_i(t) \leq t \tag{5.13}$$

Proof. *The "only if" part is proven by the definition $\widehat{W}_i(\widehat{R}_i) = \widehat{R}_i$. For the "if" part, we can use the same approach as above. Let t^* once more denote the critical scheduling point*

of the approximate testing set $\widehat{\mathcal{S}}_i$. Let us consider the ordered scheduling points $\{t_{ij}, ..., t_{il}\}$. Then, assuming $t^* = t_{ij}$, $\forall t \in \{t_{i1}, ..., t_{ij-1}\}$, we have $\widehat{W}_i(t) > t$ and $\forall t \in \{t_{ij}, ..., t_{il}\}$, thus $\widehat{W}_i(t) \leq t$.

However, since the approximate cumulative request-bound function is not a step function any more, but instead could have a positive slope between two scheduling points due to the approximation, we cannot use the same assumption as above. Specifically, if t is the time between $(t_{ij-1}, t_{ij}]$ which intersects the line given by t, it does not mean that $\widehat{W}_i(t) = \widehat{W}_i(t^*)$. However, since both $\widehat{W}_i(t)$ and t are positively increasing, there exists only one point between $(t_{ij-1}, t_{ij}]$ which corresponds to the intersection $\widehat{W}_i(t) = t$, thus $\forall t < \widehat{R}_i$, then $\widehat{W}_i(t) > t$. Therefore, if $\exists t \in \widehat{\mathcal{S}}_i : \widehat{W}_i(t) \leq t$, then it implies that for this time point we have $t \geq \widehat{R}_i$, and since the testing set is limited from above by the deadline D_i, then we have that $\widehat{R}_i \leq D_i$.

2 Lemma. *We are now able to bound the approximate worst-case response time. For a task τ_i then:*

$$R_i \leq \widehat{R}_i \leq \left(\frac{k+1}{k}\right) R_i \tag{5.14}$$

Proof. *By lemma 2, we know that $W_i(\widehat{R}_i) \leq \widehat{W}_i(\widehat{R}_i)$. For a fully available processor we know that $\widehat{W}_i(\widehat{R}_i) = \widehat{R}_i$, therefore $R_i \leq \widehat{R}_i$, since the intersection point of the exact cumulative request-bound function lies at the same point or before the intersection point of the approximate request-bound function.*

Furthermore the lemma give us also that $\widehat{W}_i(\frac{k+1}{k}R_i) \leq \frac{k+1}{k}W_i(\frac{k+1}{k}R_i)$. The function $\frac{k+1}{k}W_i(\frac{k+1}{k}R_i)$ can be seen as the execution of the task with its worst-case execution time augmented by $\frac{k}{k+1}$ (see [15] and [26]). The worst-case response analysis gives therefore $\frac{k+1}{k}W_i(\frac{k+1}{k}R_i) = \frac{k+1}{k}R_i$, which by lemma 1 gives $\widehat{R}_i \leq \frac{k+1}{k}R_i$

To conclude, we can summarise the results in the following theorem:

Theorem (RTA Bound). *The approximate worst-case response bound $\widehat{R}_i$ is on the one side an upper bound for the exact response time R_i and, on the other side, a lower bound for the worst-case response time for a $\frac{k}{k+1}$-capacity processor. The proof follows from equation 5.14 of the lemma above.*

5.3 FPTAS for Periodic Task with Jitter

Moving on to other task models, a periodic task with jitter is defined with a parameter j denoting the deviance of the task releasing time w.r.t. the time axis. Most of the literature in classical scheduling analysis tends to bound the jitter to the period of the task. In such a case the RBF is defined as follows:

$$RBF_{pj}(\tau_i, t) := \left\lceil \frac{t + j_i}{p_i} \right\rceil e_i \tag{5.15}$$

Intuitively, the jitter produces in the worst case a shifting of j_i of the task release time. In real-time calculus, this pattern is represented by the following equation for the arrival curves:

$$\alpha^u(\Delta) := \left\lceil \frac{\Delta + j_i}{p_i} \right\rceil e_i \tag{5.16}$$

Following the work of Fisher and Baruah in [15], we could use the same approach in order to derive an FPTAS for this particular task model. The approximation scheme takes as input a value ϵ, $0 < \epsilon < 1$, by which the computation accuracy can be defined. Through the parameter k we are able to define the approximate request-bound function as:

$$\widehat{RBF}_{pj}(\tau_i, t) = \begin{cases} RBF_{pj}(\tau_i, t) & \text{if } t \leq (k-1)p_i - j_i \\ (t + p_i + j_i)\frac{e_i}{p_i} & \text{otherwise} \end{cases} \tag{5.17}$$

Similarly, the equation for the approximate cumulative request-bound function is given by:

$$\widehat{W}_i(t) := e_i + \sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}_{pj}(\tau_j, t) \tag{5.18}$$

Correctness of the Approximation

Property (1 - pj). *$\forall t \geq 0$, $\widehat{RBF}_{pj}(\tau_i, t) \geq RBF_{pj}(\tau_i, t)$*

Proof. *For all $t \in (0, (k-1)p_i - j_i]$, then by definition we have that $\widehat{RBF}_{pj}(\tau_i, t) = RBF_{pj}$.*

For $t > (k-1)p_i - j_i$ then we need to prove that $(t + p_i + j_i)\frac{e_i}{p_i} \geq \left\lceil \frac{t+j_i}{p_i} \right\rceil e_i$. That is true because of the relaxation of the ceiling function, i.e $\widehat{RBF}_{pj}(\tau_i, t) = (t + p_i + j_i)\frac{e_i}{p_i} = \left(\frac{t+j_i}{p_i} + 1\right) e_i \geq \left\lceil \frac{t+j_i}{p_i} \right\rceil e_i$.

Property (2 - pj). *If $\widehat{RBF}_{pj}(\tau_i, t) > RBF_{pj}(\tau_i, t)$, then we have calculated at least $k-1$ steps of the arrival curve, i.e. $RBF_{pj}(\tau_i, t) \geq ke_i$*

Proof. *If we have $\widehat{RBF}_{pj}(\tau_i, t)$ strictly bigger than $RBF_{pj}(\tau_i, t)$, it implies that $t > (k-1)p_i - j_i$. Hence, $RBF_{pj}(\tau_i, t) = \left\lceil \frac{t+j_i}{p_i} \right\rceil e_i > \left\lceil \frac{(k-1)p_i - j_i + j_i}{p_i} \right\rceil e_i = (k-1)e_i$.*

Property (3 - pj). *$\forall t$, then the approximation never exceeds the arrival curve by more than one step size, i.e. $\widehat{RBF}_{pj}(\tau_i, t) - RBF_{pj}(\tau_i, t) \leq e_i$.*

Proof. *Again, if $t \leq (k-1)p_i - j_i$, then $\widehat{RBF}_{pj}(\tau_i, t) = RBF_{pj}(\tau_i, t)$ and thus $\widehat{RBF}_{pj}(\tau_i, t) - RBF_{pj}(\tau_i, t) = 0 \leq e_i$.*

For $t > (k-1)p_i - j_i$ we have that $\widehat{RBF}_{pj}(\tau_i, t) - RBF_{pj}(\tau_i, t) = \left(\frac{t+j_i}{p_i}\right) e_i + e_i - \left\lceil \frac{t+j_i}{p_i} \right\rceil e_i \leq \left(\frac{t+j_i}{p_i}\right) e_i + e_i - \frac{t+j_i}{p_i} e_i = e_i$.

Property (4 - pj). *The last property shows that the approximated function is bounded above by the request-bound function multiplied by the factor* $\frac{k+1}{k}$*, that is* $\forall t \geq 0$*,* $RBF_{pj}(\tau_i, t) \leq \widehat{RBF}_{pj}(\tau_i, t) \leq \left(\frac{k+1}{k}\right) RBF_{pj}(\tau_i, t)$*.*

Proof. *For* $t \leq (k-1)p_i$*, then* $\widehat{RBF}_{pj}(\tau_i, t) = RBF_{pj}(\tau_i, t)$*, since* $k \geq 1$*, for all* $0 < \epsilon < 1$ *then* $\left(\frac{k+1}{k}\right) > 1$ *and thus the relation holds.*

If $t > (k-1)p_i$*, therefore by property 1 -pj we have that* $RBF_{pj}(\tau_i, t) \leq \widehat{RBF}_{pj}(\tau_i, t)$*. For property 3 -pj we know that* $\widehat{RBF}_{pj}(\tau_i, t) \leq RBF_{pj}(\tau_i, t) + e_i$ *and for property 2 -pj we get* $RBF_{pj}(\tau_i, t) + e_i \leq RBF_{pj}(\tau_i, t) + \frac{RBF_{pj}(\tau_i, t)}{k}$*. This proves the statement.*

Following the same verification scheme adopted in [15], we can now proceed to define a series of lemmas for the considered model.

Lemma (1 - pj). *If* $\widehat{W}_i(t) \leq t$*, then* $W_i(t) \leq t$*.*

Proof. *By property 1 -pj, the summation of the approximated request-bound function is bigger than or equal to the sum of the exact request-bound function of all higher priority tasks. Formally,* $\sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}_{pj}(\tau_j, t) \geq \sum_{\tau_j \in hp(\tau_i)} RBF_{pj}(\tau_j, t)$*. Furthermore, the definitions of* $W_i(t)$ *and* $\widehat{W}_i(t)$ *give us* $e_i + \sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}_{pj}(\tau_j, t) \leq t \Rightarrow e_i + \sum_{\tau_j \in hp(\tau_i)} RBF_{pj}(\tau_j, t) \leq t$*, which proves the lemma.*

The following lemma bounds from below the error given by the approximation. If the approximate cumulative request-bound function lies above the line $f(t) = t$, then the exact cumulative request-bound function must lie above the line $f(t) = \frac{k}{k+1}t$.

Lemma (2 - pj). *If* $\widehat{W}_i(t) > t$*, then* $W_i(t) > \frac{k}{k+1}t$*.*

Proof. *By definition,* $\widehat{W}_i(t) = e_i + \sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}_{pj}(\tau_j, t) > t$*. Using property 4 -pj, this implies that* $e_i + \sum_{\tau_j \in hp(\tau_i)} \left(1 + \frac{1}{k}\right) RBF_{pj}(\tau_j, t) \Rightarrow \left(1 + \frac{1}{k}\right) \left(e_i + \sum_{\tau_j \in hp(\tau_i)} RBF_{pj}(\tau_j, t)\right) > t$*, which gives* $\left(e_i + \sum_{\tau_j \in hp(\tau_i)} RBF_{pj}(\tau_j, t)\right) > \frac{k}{1+k}t$*.*

We are now able to prove that i) if there exists a point between the task releasing time and its deadline, where the approximate cumulative request-bound function lies below the line $f(t) = t$, then the task can be scheduled under the original task set. Otherwise, the task is not feasible under a processor with $(1 - \epsilon)$-times the original processor speed. In this way, we are able to prove theorems 2 and 3 for the periodic task with jitter model.

Proof (Theorem 2 - pj). *Recall theorem 2, it states that if* $\exists t \in (0, D_i] : \widehat{W}_i(t) \leq t$*, then the task set is schedulable under DM. Let us assume a* $t_0 \in (0, D_i]$*, for which* $\widehat{W}_i(t_0) \leq t_0$*. That means, by lemma 1 - pj that* $W_i(t_0) \leq t_0$*, hence for theorem 1 the task set is schedulable.*

Proof (Theorem 3 - pj). *The theorem originally describes a lower bound for the infeasibility of the task considered, i.e. if* $\forall t \in (0, D_i] : \widehat{W}_i(t) > t$*, then the task* τ_i *is infeasible on a processor of* $(1-\epsilon)$ *capacity. The proof is derived by contradiction. Let us assume that* $\forall t \in (0, D_i], \widehat{W}_i(t) > t$*, but the task is still feasible on a* $(1-\epsilon)$ *slower processor. From 5.4, we have that* $1 - \frac{k}{k+1} \leq \epsilon$*, i.e. for theorem 1 there is a* $t_0 \in (0, D_i]$ *so that* $W_i(t_0) \leq (1-\epsilon)t_0 \leq \left(\frac{k}{k+1}\right) t_0$*. However, for lemma 2 - pj we have that if* $\widehat{W}_i(t_0) > t_0$ *then* $W_i(t_0) > \frac{k}{k+1} t_0$*, which is a contradiction, hence task* τ_i *is infeasible on a* $(1-\epsilon)$ *processor.*

5.3.1 Testing Set for Periodic Tasks with Jitter

As we mentioned above, in our PJD* component we consider sets of tasks with constrained deadlines and jitters; in this way, we simplify the analysis to only the first job of a task. For an exact analysis the testing set of a periodic with jitter task model, where $0 \leq j_i \leq D_i$ for a task τ_i, is defined by:

$$\mathcal{S}_i := \left\{ t = bp_a - j_a : a = 1, ..., i; b = 1, ..., \left\lfloor \frac{D'_i + j_a}{p_a} \right\rfloor \right\} \cup \left\{ D'_i \right\} \tag{5.19}$$

with D'_i denoting $D'_i = D_i - j_i$ The elements contained in this set depend again on the tasks' periods, that give a pseudo-polynomial number. Using the approximation factor ϵ we can bound the number of elements of the set to the value of k [31]:

$$\widehat{\mathcal{S}}_i := \{ t = bp_a - j_a : a = 1, ..., i-1; b = 1, ..., k-1 \} \cup \{ t = D_i - j_i \} \tag{5.20}$$

The sufficiency of the testing set for the periodic task with jitter model can be easily proven as the one for the periodic task model.

5.3.2 RTA for Periodic Tasks with Jitter

For the periodic task with jitter model, we need to take into account the deviance given by the jitter. In this case, as Richard et al. in [31] as well as Chen et al. in [10] show, the worst-case response time analysis is similar to that of periodic tasks, but since the jitter produces a shifting in the arrival time of a job task, this job has to wait at most j_i units of time before being processed, therefore the wcrt is given by:

$$R_i = W_i(t) + j_i \tag{5.21}$$

The proof that the first testing point in the set $\mathcal{S}_i$ which satisfies $W_i(t) \leq t$, is related to R_i like in the above equation 5.21 and is conceptually similar to the one shown in theorem 5.2.6, therefore we do not report it.

Similarly, we can define the bounds for the approximate worst-case response time of a task τ_i. The proof is basically the same as in the case of strictly periodic tasks, however we need to take into account the jitter deviance j_i both in the exact and in the approximate case, but the analysis still remains the same.

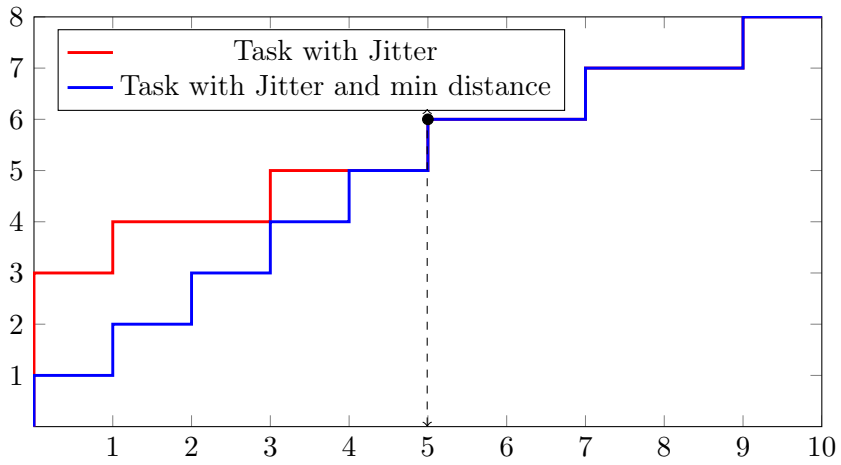

Figure 5.2: Comparison between a task with jitter and with an additional minimum inter-arrival distance.

Formally, the approximate worst-case response time for a periodic task with jitter is given by:

$$\widehat{R}_i := \min\{t > 0 : \widehat{W}_i(t) = t\} + j_i \tag{5.22}$$

Since we add to every element of the inequality 5.14 the term j_i, all the proofs still hold.

5.4 FPTAS for Periodic Task with Burst

Real-time calculus widely uses a particular arrival pattern for the incoming tasks: the so called *periodic task with jitter and minimum inter-arrival time*. This pattern was firstly presented by Richter in his PhD-thesis [32]. Traditionally, the minimum inter-arrival time of a task is given by its period. However, in cases where the jitter exceeds the period of the task, it is often necessary to reduce its influence on incoming events.

For example, in a task with period $p = 2$ and jitter $j = 5$, we could have up to 3 incoming events at the same time (see figure 5.2). Nonetheless, if, for example, the jobs are delivered by some type of sensor, it may be impossible to have all of them arriving at the same time. Instead, they can be generated at most within a minimum interval between them. For this reason, parameter d is introduced, and with it the concept of *burst*. A burst corresponds to a time interval where the sporadic incoming events are nevertheless bounded.

Let us denote the pjd task model as a periodic task τ_i with a period p_i, a jitter j_i and a minimum inter-arrival distance d_i, plus the computation time e_i. The value of d_i is bounded from above by the period, i.e. $d_i < p_i$. The behaviour of such a task is represented in real-time calculus by the following equation:

$$\alpha^u(\Delta) = \min\left(\left\lceil \frac{\Delta + j_i}{p_i} \right\rceil, \left\lceil \frac{\Delta}{d_i} \right\rceil\right) e_i \tag{5.23}$$

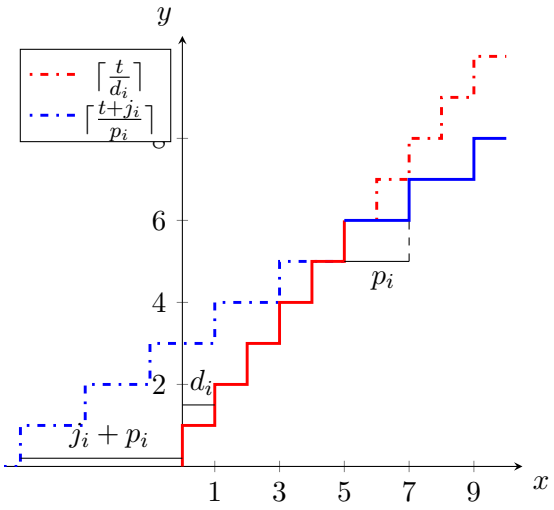

Figure 5.3: Request-bound function (solid line) for a pjd task.

This event model is widely used in the RTC literature, for example by Wandeler in [42] and by Künzli in [19]. In order to reduce its computation effort, we could extend the same approach used by Fisher and Baruah in [14] so that we can derive an approximation scheme for our PJD* component. The main idea is to *relax* the ceiling function for the *pjd* task after k steps of the upper arrival curve. Nevertheless, since now the request-bound function is defined trough a *minimum* function, we need to carefully consider the point where the approximation takes place, as the model is now described by two functions.

The request-bound function for a periodic task with jitter and minimum inter-arrival time is defined by which one of the two functions $\left\lceil \frac{t+j_i}{p_i} \right\rceil$ and $\left\lceil \frac{t}{d_i} \right\rceil$ dominates the other. In figure 5.3, we can see how the two functions are represented. The dash line denotes that the function is dominating the other, the solid line represents when the function is dominated and corresponds to the request-bound function. Formally:

$$RBF_{pjd}(\tau_i, t) := \min\left(\left\lceil \frac{t+j_i}{p_i} \right\rceil, \left\lceil \frac{t}{d_i} \right\rceil\right) e_i \tag{5.24}$$

Again, the ϵ-parameter as defined in equation 5.4 denotes the error tolerated by our approximation and gives us the $k = \lceil 1/\epsilon \rceil - 1$ number of steps for which we calculate the exact response-bound function.

As already shown above, the behaviour of the request-bound function is now described by two functions defining two distinctive regions: the *burst-region* and the *periodic with jitter region*.

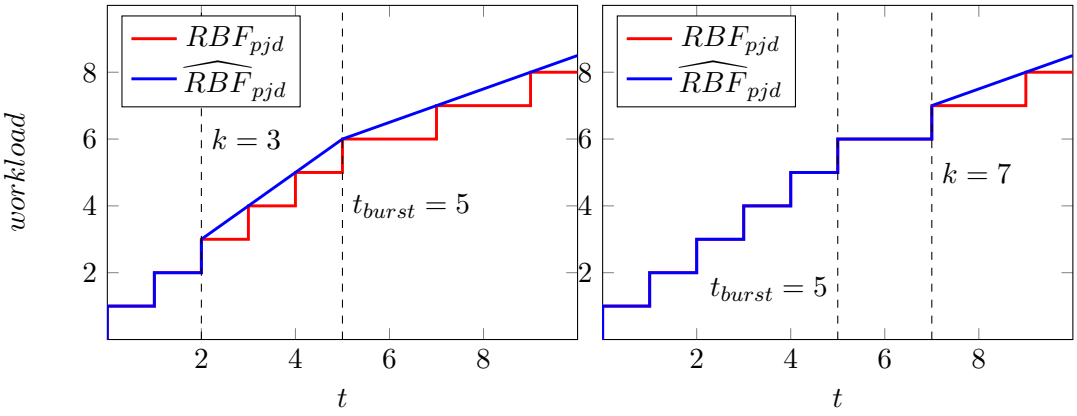

Figure 5.4: Approximation Point before and after t_{burst}.

Intuitively, the approximate RBF given by the relaxation of the ceiling function can be described by $(t + d_i)\frac{e_i}{d_i}$ or $(t + p_i + j_i)\frac{e_i}{p_i}$ depending on which function dominates the other 5.4. Formally, the approximate request-bound function is now given by:

$$\widehat{RBF}_{pjd}(\tau_i, t) = \begin{cases} \text{if } \left\lceil \frac{t}{d_i} \right\rceil < \left\lceil \frac{t+j_i}{p_i} \right\rceil & \text{then } RBF_{pjd}(\tau_i, t), \text{ for } t \leq (k-1)d_i \\ & (t + d_i)\frac{e_i}{d_i} \text{ otherwise} \\ \text{if } \left\lceil \frac{t+j_i}{p_i} \right\rceil \leq \left\lceil \frac{t}{d_i} \right\rceil & \text{then } RBF_{pjd}(\tau_i, t), \text{ for } t \leq (k-1)p_i - j_i \\ & (t + p_i + j_i)\frac{e_i}{p_i} \text{ otherwise} \end{cases} \tag{5.25}$$

Nevertheless, an important property considered in the previous case is now missing. The burst region represents in fact a periodic task with arbitrary deadlines model. The period of the task is now given by the minimum inter-arrival distance between two tasks, nevertheless since in the other models we assumed implicit deadlines, i.e. $p_i = D_i$, we have now arbitrary deadlines in this region. In this particular case the critical instant of a task set is not depicted any more by the simultaneous release of a job for every task, as Lehoczky demonstrated in [20].

That means, we can no longer base our feasibility test only on the first job of a task, since the worst-case response time could also take place in a later release. The request-bound function remains the same, however the cumulative request-bound function has now to consider more than the first job.

In fact, for a time point x, the cumulative request-bound function is now given by:

$$W_i(l, x) := \min_{t \leq x} \left(\frac{le_i + \sum\limits_{\tau_j \in hp(\tau_i)} RBF_{pjd}(\tau_j, t)}{t} \right) \tag{5.26}$$

Thus leading to an approximate request-bound function defined as follows:

$$\widehat{W}_i(l,x) := \min_{t \leq x} \left(\frac{le_i + \sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}_{pjd}(\tau_j, t)}{t} \right) \tag{5.27}$$

Therefore, theorem 1 is no longer valid. Lehoczky presents in [20] a method for analysing the schedulability of a set of tasks with arbitrary deadlines. The number of jobs that has to be considered is bounded by the *level-i* busy period. A level-i busy period is a time interval within which only jobs of priority equal or bigger than i are being executed. Moreover, Lehoczky proves that the longest worst-case response time happens within the level-i busy period initiated by the simultaneous release of a job for every task of priority i or higher (see [20, p.203]).

For this reasons, the schedulability test must meet two conditions. Let us consider the first job release of task τ_i. On the one side, we need to check if there exists a time point $\in (0, D_i]$ such as $W_i(1, D_i) \leq 1$. That means, $\tau_{i,l}$ will meet its deadline. On the other side, we also need to verify if the level-i busy period has terminated. That means, there exists a time point $\in (0, p_i]$ such as $W_i(1, p_i) \leq 1$. If such a point exists, then the level-i busy period has been completed and we can stop our computation. If there is no such point, then it means that the level-i busy period continues with the second job release of task τ_i. As Lehoczky explains, for the second job we need to verify the following inequalities: $W_i(2, p_i + D_i) \leq 1$ and $W_i(2, 2p_i) \leq 1$.

Let us denote with $\mathcal{N}_i$ the number of jobs contained by the busy period. This number is given by the following equation:

$$\mathcal{N}_i = \min_l \{l : W_i(l, lp_i) \leq 1\} \tag{5.28}$$

Similarly, the number of jobs in the approximate level-i busy period is defined as:

$$\widehat{\mathcal{N}}_i = \min_l \left\{ l : \widehat{W}_i(l, lp_i) \leq 1 \right\} \tag{5.29}$$

With this definition, we are able to extend theorem 1 to the case of tasks with arbitrary deadlines.

Theorem (Feasibility for Arbitrary Deadlines). *A set T of n tasks with arbitrary deadlines is feasible if and only if:*

$$\max_{1 \leq i \leq n} \max_{l \leq \mathcal{N}_i} W_i(l, (l-1)p_i - D_i) \leq 1 \tag{5.30}$$

It is clear that the complexity of the test increases with the need to check every job of a task τ_i within the level-i busy period. We will first show the correctness of the

approximation, and then how the approximate set of scheduling points can be derived. Moreover, we will show how it is not necessary to test every job within the busy period, so that the approximate feasibility test and the approximate response time analysis are again of pseudo-polynomial complexity.

Correctness of the Approximation

Since the $\widehat{RBF}_{pjd}$ function has changed, we need to prove again the four properties and the two lemmas in order to verify the correctness of the approximation. For this event model we need to carefully consider in which region the approximation starts.

Property (1 - pjd). $\forall t \geq 0$, $\widehat{RBF}_{pjd}(\tau_i, t) \geq RBF_{pjd}(\tau_i, t)$

Proof. *Let us suppose a time point t such as $\left\lceil \frac{t}{d_i} \right\rceil < \left\lceil \frac{t+j_i}{p_i} \right\rceil$. That means, the time point t lies within the burst region. In this case, if $t \leq (k-1)d_i$, then we calculate the exact request-bound function, hence $\widehat{RBF}_{pjd}(\tau_i, t) = RBF_{pjd}(\tau_i, t)$. On the other hand, we have that $t > (k-1)d_i$, so that the $\widehat{RBF}_{pjd}(\tau_i, t) = (t+d_i)\frac{e_i}{d_i} = (\frac{t_i}{d_i}+1)e_i \geq \left\lceil \frac{t}{d_i} \right\rceil e_i = RBF_{pjd}(\tau_i, t)$.*

The second case happens if we have a time point t, for which $\left\lceil \frac{t+j_i}{p_i} \right\rceil < \left\lceil \frac{t}{d_i} \right\rceil$. In the same way, if $t \leq (k-1)p_i - j_i$, then we compute the request-bound function exactly, i.e. $\widehat{RBF}_{pjd}(\tau_i, t) = RBF_{pjd}(\tau_i, t)$. If $t > (k-1)p_i - j_i$, the we have $\widehat{RBF}_{pjd}(\tau_i, t) = (t + p_i + j_i)\frac{e_i}{p_i} = \left(\frac{t+j_i}{p_i} + 1\right) e_i \geq \left\lceil \frac{t+j_i}{p_i} \right\rceil e_i = RBF_{pjd}(\tau_i, t)$, which proves the property.

Property (2 - pjd). *As shown above, the second property states that if $\widehat{RBF}_{pjd}$ is strictly larger than RBF_{pjd}, then we have already an execution demand of $k \cdot e_i$. Formally if $\widehat{RBF}_{pjd} > RBF_{pjd}$, then $RBF_{pjd} \geq ke_i$.*

Proof. *In the same way as above, we need to consider the two different situations. Let us suppose a time point t_0 for which $\left\lceil \frac{t_0}{d_i} \right\rceil < \left\lceil \frac{t_0+j_i}{p_i} \right\rceil$. If we have that $\widehat{RBF}_{pjd} > RBF_{pjd}$, then by definition $t_0 > (k-1)d_i$, otherwise the two functions would be the same. Therefore, the time point t_0 is larger than $(k-1)d_i$. The exact request-bound function is defined at this point as $RBF_{pjd}(\tau_i, t_0) = \left\lceil \frac{t_0}{d_i} \right\rceil e_i > \left\lceil \frac{(k-1)d_i}{d_i} \right\rceil e_i = (k-1)e_i$. Since RBF_{pjd} increases by e_i with every step, we have $RBF_{pjd} \geq ke_i$.*

For the other case, let us suppose that for t_1 we have $\left\lceil \frac{t_1}{d_i} \right\rceil \geq \left\lceil \frac{t_1+j_i}{p_i} \right\rceil$. Since we assume that the two request-bound functions are not equal, we have that $t_1 > (k-1)p_i - j_i$. The exact request-bound function is represented by $RBF_{pjd}(\tau_i, t_1) = \left\lceil \frac{t_1+j_i}{p_i} \right\rceil e_i > \left\lceil \frac{(k-1)p_i - j_i + j_i}{p_i} \right\rceil e_i = (k-1)e_i$.

Property (3 - pjd). *The third property bounds the approximation of $\widehat{RBF}_{pjd}$, i.e. $\forall t \geq 0$, then $\widehat{RBF}_{pjd}(\tau_i, t) - RBF_{pjd}(\tau_i, t) \leq e_i$*

Proof. *As usual we consider the two cases. For a time point t_0 such that $\left\lceil \frac{t_0}{d_i} \right\rceil < \left\lceil \frac{t_0+j_i}{p_i} \right\rceil$. If $t_0 \leq (k-1)d_i$, then it implies that $\widehat{RBF}_{pjd}(\tau_i, t_0) - RBF_{pjd}(\tau_i, t_0) = 0 \leq e_i$, since*

the two RBF_{pjd} *are the same function for* t_0*. On the other hand, if* $t_0 > (k-1)d_i$*, then* $\widehat{RBF}_{pjd}(\tau_i, t_0) - RBF_{pjd}(\tau_i, t_0) = (t_0 + d_i)\frac{e_i}{d_i} - \left\lceil \frac{t_0}{d_i} \right\rceil e_i \leq (t_0 + d_i)\frac{e_i}{d_i} - \frac{t_0}{d_i}e_i = e_i + \frac{t_0 e_i}{d_i} - \frac{t_0 e_i}{d_i} = e_i$*.*

If for a time point t_i *we have* $\left\lceil \frac{t_1}{d_i} \right\rceil \geq \left\lceil \frac{t_1 + j_i}{p_i} \right\rceil$*, we need to consider as well, whether the two request-bound function are the same or a different function. For the first situation,* $t_1 \leq (k-1)p_i - j_i$ *so that* $\widehat{RBF}_{pjd}(\tau_i, t_1) - RBF_{pjd}(\tau_i, t_1) = 0 \leq e_i$*. If, on the other hand,* $t_1 > (k-1)p_i - j_i$*, then* $\widehat{RBF}_{pjd}(\tau_i, t_1) - RBF_{pjd}(\tau_i, t_1) = (t_1 + p_i + j_i)\frac{e_i}{p_i} - \left\lceil \frac{t_1 + j_i}{p_i} \right\rceil e_i \geq (t_1 + p_i + j_i)\frac{e_i}{p_i} - \frac{t_1 + j_i}{p_i}e_i = e_i + \frac{(t_1 + j_i)e_i}{p_i} - \frac{t_1 + j_i}{p_i}e_i = e_i$*.*

Property (4 - pjd). *Finally the fourth property allows us to bound from below the approximation of the exact function. Formally:* $\forall t \geq 0$*,* $RBF_{pjd}(\tau_i, t) \leq \widehat{RBF}_{pjd}(\tau_i, t) \leq \left(\frac{k+1}{k}\right) RBF_{pjd}(\tau_i, t)$*.*

Proof. *For the first case, where given a time point* t_0 *for which* $\left\lceil \frac{t_0}{d_i} \right\rceil < \left\lceil \frac{t_0 + j_i}{p_i} \right\rceil$*, we consider at first if* $t_0 \leq (k-1)d_i$*, then* $\widehat{RBF}_{pjd}(\tau_i, t_0) = RBF_{pjd}(\tau_i, t_0)$*. Since* $k \geq 1$*, for* $0 < \epsilon < 1$*, we have* $\frac{k+1}{k} > 1$*, therefore* $RBF_{pjd}(\tau_i, t_0) = \widehat{RBF}_{pjd}(\tau_i, t_0) \leq (\frac{k+1}{k}) RBF_{pjd}(\tau_i, t_0)$*. For* $t_0 > (k-1)d_i$*, then by property 1 - pjd we know that* $RBF_{pjd}(\tau_i, t_0) \leq \widehat{RBF}_{pjd}(\tau_i, t_0)$*. Property 3 - pjd gives* $\widehat{RBF}_{pjd}(\tau_i, t_0) \leq RBF_{pjd}(\tau_i, t_0) + e_i$*. Finally property 2 - pjd implies* $RBF_{pjd}(\tau_i, t_0) + e_i \leq RBF_{pjd}(\tau_i, t_0) + \frac{RBF_{pjd}(\tau_i, t_0)}{k}$*, and therefore* $\widehat{RBF}_{pjd}(\tau_i, t_0) \leq (\frac{k+1}{k}) RBF_{pjd}(\tau_i, t_0)$*.*

The case for which $\left\lceil \frac{t_1}{d_i} \right\rceil \geq \left\lceil \frac{t_1 + j_i}{p_i} \right\rceil$ *is similarly proven.*

We now need to modify lemmas 1 - pj and 2 - pj in order to extend the results to the pjd model.

Lemma (1 - pjd). *If* $\widehat{W}_i(l, t) \leq t$*, then* $W_i(l, t) \leq t$

Proof. *The proof is straightforward and resembles the one for lemma 1 - pj. Indeed, by definition we have* $\widehat{W}_i(l, t) = le_i + \widehat{RBF}_{pjd}(\tau_i, t)$*, which we prove to be* $\geq le_i + RBF_{pjd}(\tau_i, t) = W_i(l, t)$*.*

Lemma (2 - pjd). *If* $\widehat{W}_i(l, t) > t$*, then* $W_i(l, t) > \frac{k}{k+1}t$

Proof. *From property 4 - pjd, we know that if* $le_i + \widehat{RBF}_{pjd}(\tau_i, t) > t$*, then* $le_i + RBF_{pjd}(1 + \frac{1}{k})(\tau_i, t) \Rightarrow (1 + \frac{1}{k})\left(le_i + RBF_{pjd}(\tau_i, t)\right) > t$ *since* $(1 + \frac{1}{k})$ *is a positive value. Therefore we conclude* $(le_i + RBF_{pjd}(\tau_i, t)) = W_i(l, t) > (\frac{k}{1+k})t$*.*

5.4.1 Testing Set for Period Tasks with Burst

As in the previous models, we define a testing set for the periodic task with burst FPTAS. Since we are using a minimum function to define the request-bound function, the testing set has to take into account both the burst region and the jitter region.

The testing set of a task τ_i also depends on the job ending the busy period. However, we do not known if the busy period ends within the burst region or the jitter region.

Let us denote with B the smallest job release which ends the busy period, therefore we could have the following situation:

1. if the busy period ends within the burst region, then:

$$W_i(l,t) \leq ld_i \tag{5.31}$$

2. instead, if the busy period ends within the jitter region, then:

$$W_i(l,t) \leq lp_i - j_i \tag{5.32}$$

We can notice that the smallest job l which satisfies equation 5.31 must satisfy equation 5.32 as well. Therefore, we can define B as:

$$B := \min_l(W_i(l,t) \leq ld_i) \tag{5.33}$$

Therefore, the exact testing set can be formally define it as:

$$\begin{aligned} \mathcal{S}_i := \{t = bd_a : a = 1, ..., i; b = 1, ..., B\} \cup \\ \{t = bp_a - j_a : a = 1, ..., i; b = 1, ..., B\} \end{aligned} \tag{5.34}$$

For the approximate testing set, we first define the value of $t_{i,burst}$, which represents the point delimiting the two approximate curves:

$$t_{i,burst} = \frac{j_i d_i}{p_i - d_i} \tag{5.35}$$

The number of steps in both regions is delimited by the value of $\left\lceil \frac{t_{a,burst}}{d_a} \right\rceil - 1$, which gives the number of jobs releases by a task τ_a within the burst region. The first job released after $t_{a,burst}$ is released at time $\left\lceil \frac{t_{a,burst}}{d_a} \right\rceil p_a - j_a$.

Also in this case, the approximation factor ϵ defines the number of steps exactly computed. However, the point where the actual steps occur depends again on which of the two functions is dominated. Moreover we will show later in the chapter why the point $t_{i,burst}$ is important:

$$\begin{aligned} \widehat{\mathcal{S}}_i := \left\{t = bd_a : a = 1, ..., i-1; b \in \mathbb{N} : b \leq (k-1) \wedge b \leq \left\lceil \frac{t_{a,burst}}{d_a} \right\rceil - 1\right\} \cup \\ \left\{t = bp_a - j_a : a = 1, ..., i-1; b \in \mathbb{N} : \left\lceil \frac{t_{a,burst}}{d_a} \right\rceil \leq b \wedge b \leq (k-1)\right\} \cup \{t_{i,burst}\} \end{aligned} \tag{5.36}$$

Nevertheless, since now the approximate cumulative request-bound function depends also on the number of jobs within the level-i busy period, the approximation scheme introduced above is not of polynomial complexity any more. In fact, as Fisher points out

in [14], the number of jobs that has to be consider within the level-i busy period does not depend on the value of n and ϵ. The exact length of the busy period can be found by solving the following equation:

$$t = \sum_{j=1}^{i} \min\left(\left\lceil \frac{t + j_i}{p_i} \right\rceil, \left\lceil \frac{t}{d_i} \right\rceil\right) e_i \tag{5.37}$$

It is clear that the number of jobs within the level-i busy period leads to a pseudo-polynomial complexity. Nevertheless, we can define an approximate analysis in order to limit the complexity and to derive an FPTAS for the periodic with burst task model.

We begin by defining the approximate response time for our task model. In this case, the intersection point $R_{i,l}$ between the cumulative request-bound function for the l^{th} job and the line given by $f(t) = t$ does not describe a response time any more, since the job could have also been released at a later time than 0. Moreover, the burst and jitter regions differentiate the definition as follows:

$$RT_{i,l_{burst}} := R_{i_l} - (l - i)d_i \tag{5.38}$$

$$RT_{i,l_{jitter}} := R_{i_l} - ((l - i)p_i - j_i) \tag{5.39}$$

Consequently, for the approximate response time we have:

$$\widehat{RT}_{i,l_{burst}} := \widehat{R}_{i_l} - (l - i)d_i \tag{5.40}$$

$$\widehat{RT}_{i,l_{jitter}} := \widehat{R}_{i_l} - ((l - i)p_i - j_i) \tag{5.41}$$

We will denote simply with $\widehat{R}_{i,l}$ if the job type does not matter in the analysis. The approximate worst-case response time for the approximate level-i busy period containing $\widehat{\mathcal{N}}_i$ is therefore defined as:

$$\widehat{RT}_i := \max_{l \leq \widehat{\mathcal{N}}_i} \widehat{RT}_{i,l} \tag{5.42}$$

Reducing the number of tested jobs

In order to reduce the complexity of the approximation scheme, we can adapt the analysis of Nguyen et al. in [26] to our task model. In this paper it is proven that the derivation of the worst-case execution time of a task can be done by analysing the intervals defined by two adjacent testing points of the set $\widehat{\mathcal{S}}_i$. Between these points the longest response time is always given by the first job of the task τ_i completed within the interval. Moreover, checking if the last job released by task τ_i completes within the interval returns if the level-i busy period has terminated or not.

3 Lemma. *Let us assume two adjacent scheduling points t_1, t_2 in $\widehat{\mathcal{S}}_i$ such as $t_1 < t_2$. For two jobs l, h of task τ_i, such as $l < h$, that have their intersection points $\widehat{R}_{i_l}, \widehat{R}_{i_h}$ lying between the interval $(t_1, t_2]$, we then have that:*

$$\widehat{RT}_{i,l} \leq D_i = p_i \Rightarrow \widehat{RT}_{i,h} \leq D_i = p_i \tag{5.43}$$

Moreover, we can define a lemma for verifying that the level-i busy period terminates within two adjacent points t_1, t_2. If the last job h of task τ_i completed its execution before the release of another job of task τ_i, then the level-i busy period terminates within $(t_1, t_2]$. We will at first consider only the jobs within the burst region.

4 Lemma. *For two adjacent scheduling points t_1, t_2 in $\widehat{\mathcal{S}}_i$ such as $t_1 < t_2$ and for two job releases l, h of task τ_i, such as $l < h$, then we have that:*

$$\widehat{RT}_{i,h} > d_i \Rightarrow \widehat{RT}_{i,l} > d_i \tag{5.44}$$

Proof. *The proof of lemmas 3 and 4 is equivalent to showing that $\widehat{RT}_{i,l} > \widehat{RT}_{i,h}$. The value of $\frac{e_j}{d_j}$ represents the utilisation of the task τ_j in the burst region. We shall denote this utilisation with $U_{j,burst}$. Moreover, at the time point t_1 there could also be tasks for which the approximation has already started, i.e. for task τ_j such as $j < i$, we have that either $(k-1)d_j \leq t_1$ or $(k-1)p_j - j_j \leq t_1$. We denote with $\mathcal{T}_{burst}$ and $\mathcal{T}_{jitter}$ the sets of tasks whose approximation has already started at time t_1 and whose approximate request-bound function is described by $(t + d_j)\frac{e_j}{d_j}$ and $(t + p_j + j_j)\frac{e_j}{p_j}$ respectively. Then, for the l^{th} job release we can write the approximate cumulative request-bound function for a time point $t \in (t_1, t_2]$ as:*

$$\begin{aligned}
\widehat{W}_i(l,t) &= le_i + \sum_{\tau_j \in hp(i)} \widehat{RBF}_{pjd}(\tau_j, t) \\
&= le_i + \sum_{\tau_j \notin \{\mathcal{T}_{burst} \vee \mathcal{T}_{jitter}\}} RBF_{pjd}(\tau_j, t) + \sum_{\tau_j \in \mathcal{T}_{burst}} (t + d_j)U_{j,burst} + \sum_{\tau_j \in \mathcal{T}_{jitter}} (t + p_j + j_j)U_j \\
&= le_i + \sum_{\tau_j \notin \{\mathcal{T}_{burst} \vee \mathcal{T}_{jitter}\}} RBF_{pjd}(\tau_j, t) + \sum_{\tau_j \in \mathcal{T}_{burst}} d_j U_{j,burst} + \sum_{\tau_j \in \mathcal{T}_{jitter}} (p_j + j_j)U_j + \\
&+ t \left(\sum_{\tau_j \in \mathcal{T}_{burst}} U_{j,burst} + \sum_{\tau_j \in \mathcal{T}_{jitter}} U_j \right)
\end{aligned} \tag{5.45}$$

Let us A and B denote:

$$\begin{aligned} A &= le_i + \sum_{\tau_j \notin \{\mathcal{T}_{burst} \vee \mathcal{T}_{jitter}\}} RBF_{pjd}(\tau_j, t) + \sum_{\tau_j \in \mathcal{T}_{burst}} d_j U_{j,burst} + \sum_{\tau_j \in \mathcal{T}_{jitter}} (p_j + j_j) U_j \\ B &= \sum_{\tau_j \in \mathcal{T}_{burst}} U_{j,burst} + \sum_{\tau_j \in \mathcal{T}_{jitter}} U_j \end{aligned} \tag{5.46}$$

Therefore we have that:

$$\widehat{W}_i(l,t) = A + Bt \tag{5.47}$$

The values of A and B are constant in the interval $(t_1, t_2]$. From the definition of the approximate cumulative bound function we know that $\widehat{W}_i(h,t) = \widehat{W}_i(l,t) + (h-l)e_i$, therefore we can rewrite it as:

$$\widehat{W}_i(h,t) = (h-l)e_i + A + Bt \tag{5.48}$$

The intersection points $\widehat{R}_{i,l}$ and $\widehat{R}_{i,h}$ of $\widehat{W}_i(l,t)$ and $\widehat{W}_i(h,t)$ can therefore be rewritten as:

$$\begin{aligned} \widehat{R}_{i,l} &= \frac{A}{1-B} \\ \widehat{R}_{i,h} &= \frac{A + (h-l)e_i}{1-B} \end{aligned} \tag{5.49}$$

Thus giving us the following result:

$$\begin{aligned} \widehat{R}_{i,h} - \widehat{R}_{i,l} &= \frac{A + (h-l)e_i}{1-B} - \frac{A}{1-B} \\ &= (h-l)\frac{e_i}{1-B} \\ &= (h-l)d_i\frac{U_{i,burst}}{1-B} = RHS \end{aligned} \tag{5.50}$$

The overall utilisation U is usually smaller than one. However, in this case, since the length of d_i is strictly smaller than p_i, we could also have an utilisation $U_{burst} \geq 1$. Let us denote with T_{burst} and T_{jitter}, the tasks lying in the burst or jitter region respectively at time point t, the total utilisation is now given by $U_{i,burst} + \sum_{\tau_j \in T_{burst}} U_{j,burst} + \sum_{\tau_j \in T_{jitter}} U_j$. The original proof of Nguyen in [26] is based on the assumption that the total utilisation is smaller than one, in order to derive that $\frac{U_{i,burst}}{1-B} < 1$ and thus proving the lemmas. Nevertheless, we cannot follow the same idea since we do not know if the total utilisation is bigger or smaller than 1.

Instead, we can notice that if the total utilisation of the considered task up to τ_i is bigger than one, i.e. $U_{i,burst} + \sum_{\tau_j \in T_{burst}} U_{j,burst} + \sum_{\tau_j \in T_{jitter}} U_j > 1$, than we would have no job of task τ_i completed between the interval $(t1, t2]$, since the approximate cumulative request bound function will have no intersection point with the line $f(t) = t$, thus lemmas 3 and

4 will have no meaning in this case. In fact, if we consider the actual number of released jobs by task τ_i at time t, we have that:

$$\begin{aligned}
\widehat{W}_i(t) &\geq \left\lceil \frac{t}{d_i} \right\rceil e_i + \sum_{\tau_j \in hp(i)} RBF_{pjd}(\tau_j, t) \\
&\geq \frac{t}{d_i} e_i + \sum_{\tau_j \in T_{burst}} \frac{t}{d_j} e_j + \sum_{\tau_j \in T_{jitter}} (t + j_j) \frac{e_j}{p_j} \\
&\geq \frac{t}{d_i} e_i + t \left(\sum_{\tau_j \in T_{burst}} U_{j,burst} + \sum_{\tau_j \in T_{jitter}} U_j \right) \\
&\geq t \left(U_{i,burst} + \sum_{\tau_j \in T_{burst}} U_{j,burst} + \sum_{\tau_j \in T_{jitter}} U_j \right) \\
&> t \text{ since we assume a total utilisation bigger than one} \\
&\Rightarrow \frac{\widehat{W}_i(t)}{t} > 1
\end{aligned} \tag{5.51}$$

On the other hand, if the total utilisation at time t_1 of all the jobs $j \leq i$ is smaller than one, i.e. $U_{i,burst} + \sum_{\tau_j \in T_{burst}} U_{j,burst} + \sum_{\tau_j \in T_{jitter}} U_j \leq 1$, then we have that:

$$\begin{aligned}
1 - B &= 1 - \sum_{\tau_j \in \mathcal{T}_{burst}} U_{j,burst} - \sum_{\tau_j \in \mathcal{T}_{jitter}} U_j \\
&\geq 1 - \sum_{\tau_j \in T_{burst}} U_{j,burst} - \sum_{\tau_j \in T_{jitter}} U_j \\
&> \left(U_{i,burst} + \sum_{j \in T_{burst}} U_{j,burst} + \sum_{j \in T_{jitter}} U_j \right) - \sum_{\tau_j \in T_{burst}} U_{j,burst} - \sum_{\tau_j \in T_{jitter}} U_j \\
&= U_{i,burst} \\
&\Rightarrow \frac{U_{i,burst}}{1 - B} < 1
\end{aligned} \tag{5.52}$$

Thus, we have that $RHS < (h - l)d_i$ and therefore $\widehat{R}_{i,h} - \widehat{R}_{i,l} < (h - 1)d_i$, meaning that $\widehat{RT}_{i,l} > \widehat{RT}_{i,h}$.

To summarise the results, we have that for every two completed jobs within two adjacent scheduling points of $\widehat{\mathcal{S}}_i$, the longest response time is always given by the job released the soonest. Therefore, in order to calculate the worst-case response time of the task τ_i, we only need to test the first completed job for every interval given by two adjacent scheduling points of the testing set.

Moreover, we need to test this job only for the intervals that correspond to the level-i busy period. In addition, testing the last job completed will return if the level-i busy period terminates within this interval. In fact, given the relation above, if the last completed job terminates not later that the release of the next job, i.e. its $\widehat{RT}_{i,last} < d_i$, then also all the other jobs will finish before this time, thus the level-i busy period terminates.

To identify the index of the last job released that has been completed by time t, Nguyen in [26] defines the following lemma:

5 Lemma. *For a task τ_i, the last job completed by time t is defined as:*

$$last_i(t) = \left\lfloor \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j, t)}{e_i} \right\rfloor \tag{5.53}$$

Proof. *The proof is equivalent to showing that if the l^{th} job has been completed by time t, so did all the other jobs smaller than l. Moreover, all the jobs bigger than l will not be completed by this time:*

$$\forall l, l \leq last_i(t) \text{ then } \widehat{W}_i(l,t) \leq t$$

$$\forall l, l > last_i(t) \text{ then } \widehat{W}_i(l,t) > t$$

Let us denote with a the function $\left\lfloor \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i} \right\rfloor$. By the properties of the floor function we know that:

$$\frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i} - 1 < \left\lfloor \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i} \right\rfloor$$

$$t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t) - e_i < \left\lfloor \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i} \right\rfloor e_i$$

$$t - e_i < ae_i + \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)$$

$$t - e_i < \widehat{W}_i(a,t)$$

We have that $\forall l > a$, then $\forall l \geq a+1$, thus $\widehat{W}_i(l,t) \geq \widehat{W}_i(a,t) + e_i > t$.

Similarly, for the properties of the floor function we have that:

$$\left\lfloor \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i} \right\rfloor \leq \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i}$$

$$\left\lfloor \frac{t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)}{e_i} \right\rfloor e_i \leq t - \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t)$$

$$ae_i + \sum_{\tau_j \in hp(i)} \widehat{RBD}_{pjd}(\tau_j,t) \leq t$$

$$\widehat{W}_i(a,t) \leq t$$

Therefore $\forall l \leq a$, then $\widehat{W}_i(l,t) \leq \widehat{W}_i(a,t) \leq t$.

To summarise the analysis of the pjd model, we note that the original FPTAS presented by Fisher and Baruah in [15] was considering periodic tasks with bounded deadlines. In

their following work [14], they presented an extension in order to include arbitrary deadlines as well. To limit the necessary jobs that have to be tested, they based their assumptions on the fact that only one job can be completed within two adjacent scheduling points.

However, later Nguyen et al. in [25] demonstrate that this assumption is wrong, since more jobs can be completed within an interval. Therefore, they develop a new scheme in order to reduce the number of tests. In lemmas 3 and 4 we adapted this scheme so that it can be used within our PJD* component.

In the previous sections we have shown how to reduce the complexity of both schedulability test and response time analysis for periodic tasks and periodic tasks with jitter. This can be done by limiting the testing set as we did for $\widehat{\mathcal{S}}_i$ in the periodic, and periodic with jitter tasks model. The number of scheduling points in these sets is a polynomial number depending on the number of task i and the approximation factor k. However, this approximation scheme is fully polynomial only for constrained deadlines tasks.

We assumed for the simplest cases to have implicit deadlines, thus $p_i = D_i$. In the periodic task with burst model, we face however the problem of having arbitrary deadlines for the jobs released within the burst region, since our period is now $d_i < D_i$. In this case, even if the number of scheduling points in the approximate testing set remains polynomial, we need to check more than the first job released. The critical instant theorem, for which only the first job of a synchronous release gives the wcrt of the task, is no longer valid. As Lehoczky shows in [20], a later job of task τ_i could have a longer response time.

The number of jobs to be tested can be limited by analysing the level-i busy period. Lehoczky shows in [20] that the worst-case level-i busy period is given by the simultaneous release of a job for every task of the task set. However, the number of jobs within the level-i busy period is pseudo-polynomial. Moreover, the jobs released between two adjacent scheduling points of $\widehat{\mathcal{S}}_i$ are also a pseudo-polynomial number.

For this reasons, we introduced in our FTPAS for the pjd model the lemmas above (see 3 and 4), so we can limit the complexity of our FPTAS again. This can be done, since we have shown that the longest response time is always given by the first completed job between two adjacent scheduling points. Moreover, to verify if the busy period terminates, we only need to check if the last job completed within the interval finishes before the release of the next job of task τ_i, i.e. $\widehat{RT}_{i,last} < d_i$. In conclusion, equation 5.53 gives a simple formula in order to obtain the last completed job by time t.

However, this analysis works if the level-i busy period terminates before the last scheduling point contained in the testing set. When the level-i busy period continues beyond this point, the approximate cumulative request-bound function is described by the approximation of all higher priority tasks.

Furthermore, since in the pjd model the RBF is described by the two functions for the burst and jitter regions, we do not know which one is used after the last scheduling point of

$\widehat{\mathcal{S}}_i$ for all the higher priority tasks. For this reason, we can take into account the following observation. If we add one more point in the testing set, i.e. $t_{i,burst} = \frac{j_i d_i}{p_i - d_i}$, we have that:

1. after this point, lemmas 3 and 4 are still valid, since the proof still holds, but the verification that the level-i busy period terminates is now done by checking whether the response time of the last completed job is smaller than the period and not the minimum inter-arrival distance, hence $\widehat{RT}_{i,last} < p_i$;

2. by including $t_{j,burst}$ for all tasks $\tau_j, j \leq i$, we ensure that after the last scheduling point in $\widehat{\mathcal{S}}_i$, the cumulative approximate request-bound function is described by only the sum of the approximate request-bound functions within the jitter region.

Therefore, if the busy period does not terminate with the last scheduling point, then $\forall t > t_1 = \max(\widehat{\mathcal{S}}_i)$, we have that:

$$\widehat{W}_i(l,t) = le_i + \sum_{\tau_j \in hp(i)} (t + p_j + j_j)\frac{e_j}{p_j} \tag{5.54}$$

6 Lemma. *The longest response time of task τ_i for a time point $t > \max(\widehat{\mathcal{S}}_i)$ is given by the job released the soonest, i.e.:*

$$\forall l, h \text{ so that } l < h : \widehat{RT}_{i,l} > \widehat{RT}_{i,h} \tag{5.55}$$

Proof. *As we did in lemmas 3 and 4 we can rewrite the approximate cumulative request-bound function as:*

$$\begin{aligned}
\widehat{W}_i(l,t) &= le_i + \sum_{\tau_j \in hp(i)} (t + p_j + j_j)\frac{e_j}{p_j} \\
&= le_i + \sum_{\tau_j \in hp(i)} (p_j + j_j)U_j + t \sum_{\tau_j \in hp(i)} U_j \\
\text{Let } A &= le_i + \sum_{\tau_j \in hp(i)} (p_j + j_j)U_j \\
\text{Let } B &= \sum_{\tau_j \in hp(i)} U_j \\
&\Rightarrow \widehat{W}_i(l,t) = A + tB
\end{aligned}$$

For every $t > \max(\widehat{\mathcal{S}}_i)$, we have that A and B are constant, moreover, by definition, we have $\widehat{W}_i(h,t) = \widehat{W}_i(l,t) + (h-l)e_i$, thus yielding:

$$\widehat{W}_i(h,t) = (h-l)e_i + A + tB$$

The intersection points $\widehat{R}_{i,l}$ *and* $\widehat{R}_{i,h}$ *of the line* $f(t) = t$ *with the functions* $\widehat{W}_i(l,t)$ *and* $\widehat{W}_i(h,t)$ *respectively, is thus given by:*

$$\begin{aligned} \widehat{R}_{i,l} &= \frac{A}{1-B} \\ \widehat{R}_{i,h} &= \frac{A + (h-l)e_i}{1-B} \end{aligned} \tag{5.56}$$

Therefore we can derive that:

$$\begin{aligned} \widehat{R}_{i,h} - \widehat{R}_{i,l} &= \frac{A + (h-l)e_i}{1-B} - \frac{A}{1-B} \\ &= (h-l)\frac{e_i}{1-B} \\ &= (h-l)p_i\frac{U_i}{1-B} = RHS \end{aligned}$$

As we did above, we now need to prove that $\frac{U_i}{1-B} < 1$*. This can be shown since the total utilisation of the task set is* $U < 1$*:*

$$\begin{aligned} 1 - B &= 1 - \sum_{\tau_j \in hp(i)} U_j \\ &> \sum_{\tau_j \in T} U_j - \sum_{\tau_j \in hp(i)} U_j \\ &\geq U_i \end{aligned}$$

Thus $\frac{U_i}{1-B} < 1 \Rightarrow RHS < (h-l)p_i$*, therefore* $\widehat{R}_{i,h} - \widehat{R}_{i,l} < (h-l)p_i$*, or equivalently* $\widehat{RT}_{i,l} > \widehat{RT}_{i,h}$*.*

Consequently, if after testing all the scheduling points in $\widehat{\mathcal{S}}_i$, the level-i busy period has not yet terminated, in order to derive the longest response time after this point, we only need to test the first job that has not completed its execution by time $t_0 = \max \widehat{\mathcal{S}}_i$. This can be easily done by using formula 5.53, and by considering the next job release h of task τ_i. Moreover, after the last scheduling point, we can define the longest response-time by considering the intersection point $\widehat{W}_i(h,t) = t$, so that:

$$\begin{aligned} \widehat{R}_{i,h} &= \frac{he_i + \sum_{\tau_j \in hp(i)} \left(e_j + \frac{j_j}{p_j}\right)}{1 - \sum_{\tau_j \in hp(i)} U_j} \\ \Rightarrow \widehat{RT}_{i,h} &= \widehat{R}_{i,h} - ((h-1)p_i - j_i) \end{aligned} \tag{5.57}$$

5.4.2 RTA for Periodic Tasks with Burst

To conclude, we want to bound the approximate worst-case response time to the factor ϵ so that we can have a lower and upper bound of the task system analysis. To do so, we first prove that the intersection point between the approximate cumulative request bound function is bounded from above and below by the following property:

Property. *Let us consider a task $\tau_i \in T$, then $\forall l \in \widehat{\mathcal{N}}_i$, we have:*

$$R_{i,l} \leq \widehat{R}_{i,l} \leq \frac{k+1}{k} R_{i,l} \tag{5.58}$$

Proof. *We know that the approximate cumulative request-bound function is an upper bound for $W_{i,l}(t)$, therefore the intersection point must lie at the same point as $\widehat{W}_{i,l}(t)$ or before, so $R_{i,l} \leq \widehat{R}_{i,l}$. We also know, that if $\widehat{W}_{i,l}\left((\frac{k}{k+1})t\right) \leq \frac{k+1}{k} W_{i,l}\left((\frac{k}{k+1})t\right)$. If we consider a $\frac{k}{k+1}$ capacity processor, i.e. $\frac{k}{k+1}f(t)$, then $\frac{k+1}{k} W_{i,l}\left((\frac{k}{k+1})t\right) = \left((\frac{k}{k+1})t\right)$ which is an intersection point for the lower capacity processor, thus $\left((\frac{k}{k+1})t\right) = \frac{k+1}{k} R_{i,l}$.*

7 Lemma. *The approximate worst-case response time is bounded by the following relation:*

$$\widehat{RT}_i \leq \widehat{RT}_i \leq \left(\frac{k+1}{k}\right) RT_i \tag{5.59}$$

Proof. *We can derive from the lemma above that:*

$$RT_{i,l} \leq \widehat{RT}_{i,l} \leq \left(\frac{k+1}{k}\right) RT_{i,l} \tag{5.60}$$

Now we need to prove that the inequality still holds for every job in the level-i busy period. By definition we know that for every job in the busy period, the wcrt exceeds the minimum inter-arrival time d_i (otherwise we would not have a busy period). Let us denote with $\mathcal{N}_i$ the jobs in the exact busy period: $\forall l \in \mathcal{N}_i$ then either $RT_{i,l} > d_i$ or $RT_{i,l} > p_i + j_i$ imply $\forall l \in \mathcal{N}_i, \widehat{RT}_{i,l} > d_i$ or $\widehat{RT}_{i,l} > d_i$. Therefore the number of jobs in the approximate busy period is equal or bigger, i.e. $\mathcal{N}_i \leq \widehat{\mathcal{N}}_i$. Similarly, from equation 5.60, we know that $\forall l \in \mathcal{N}_i, \widehat{RT}_{i,l} > d_i \Rightarrow \forall l \in \mathcal{N}_i, \frac{k+1}{k} RT_{i,l} > d_i$, or for the jobs in the burst region $\forall l \in \mathcal{N}_i, \widehat{RT}_{i,l} > p_i + j_i \Rightarrow \forall l \in \mathcal{N}_i, \frac{k+1}{k} RT_{i,l} > p_i + j_i$, therefore the number of jobs in the busy period of a $\frac{k}{k+1}$-capacity processor is always equal or bigger, i.e. $\widehat{\mathcal{N}}_i \leq \frac{k+1}{k}\mathcal{N}_i$. Recalling the definition of RT_i, we then have:

$$\max_{l \leq \mathcal{N}_i} RT_{i,l} \leq \max_{l \leq \widehat{\mathcal{N}}_i} \widehat{RT}_{i,l} \leq \max_{l \leq \left(\frac{k+1}{k}\right)\mathcal{N}_i} \left(\frac{k+1}{k}\right) RT_{i,l} \tag{5.61}$$

and thus proving the lemma.

To summarise, the scheduling points in $\widehat{\mathcal{S}}_i \cup t_{i,burst}$ are sufficient to determine the worst-case response time of the pjd model. For every interval between two adjacent scheduling points t_1, t_2 of the testing set, we need to test the first completed job in order to derive the longest response time within the interval. The index of this job is given by formula 5.53, i.e. $last_i(t_1) + 1$. Moreover, to determine whether the level-i busy period terminates by time t_2, we only have to check the job with the index given by $last_i(t_2)$.

Adding the point $t_{i,burst}$ to the testing set guarantees that after analysing the last scheduling point of $\widehat{\mathcal{S}}_i$, the approximate request-bound function of all the higher priority

tasks will be described by the function $(t + p_j + j_j)\frac{e_j}{p_j}$. We proved with lemma 6 that we only need to check the job with index $last_i\,(max(\mathcal{S}_i)) + 1$.

Therefore, we can now conclude the approximate response-time analysis with the following theorem:

Theorem (RTA-pjd - Bound). *The approximate worst-case response time bound $\widehat{RT}_i$ is on the one side an upper bound for the exact response RT_i and on the other side a lower bound for the worst-case response time for a $(1-\epsilon)$-capacity processor.*

5.5 Bounded Delay Resource Model

Real-time calculus can also be used in order to model different CPU behaviours. For example, a widely used resource pattern is the so called *bounded delay*. Intuitively, such a stream represents a resource which will be available only after some time BD. This model is therefore represented by the pair (BD, B), where BD denotes the maximum delay and B is the total bandwidth of the resource. We will assume that the resource bandwidth is 100% after the initial delay.

A bounded delay resource can be represented by the following equation:

$$\beta^l(\Delta) := \max(0, \Delta - BD) \tag{5.62}$$

With a similar method as in [15] and [14], we could extend the model in order to consider this particular behaviour as well.

That means, the schedulability test for the task τ_i has to also take into account the possibility that the resource will not be available during the maximal busy window. We present now a theorem derived from the original workload analysis proposed in 1989 by Lehoczky, Sha, and Ding in [21].

Theorem (1 -Bounded Delay). *In a task model with a bounded delay BD resource availability for a fully pre-emptive task set under a fixed priority algorithm, a task τ_i is said to be feasible if*

$$\exists t \in (0, D_i], \text{ so that } W_i(t) \leq t - BD \tag{5.63}$$

Proof. *We can observe that the task τ_i finishes its computation at time $t \in (0, D_i]$ if and only if all the higher priority jobs and the computation time e_i have been completed at time t. This amount is given by the cumulative request-bound function $W_i(t)$ and is completed at time t if and only if $W_i(t) \leq t - BD$.*

In this particular case, we do not change the approximation scheme of the request-bound function. Therefore the correctness of the approximation still holds, meaning that the properties 1 - pjd, 2 - pjd, 3 - pjd, and 4 - pjd are still valid.

What we need to prove are lemmas 3 and 4 that are then used in order to extend theorems 2 and 3.

Lemma (1 - BD). *If $\widehat{W}_i(t) \leq t - BD$, then $W_i(t) \leq t - BD$.*

Proof. *This is obviously true given lemmas 1 and 1 - pjd.*

The following lemma bounds the error from below, given the approximation factor ϵ. If the approximate cumulative request-bound function lies above the line $f(t) = t$, then the exact cumulative request-bound function must lie above the line $f(t) = \frac{k}{k+1}t$.

Lemma (2 - BD). *If $\widehat{W}_i(t) > t - BD$, then $W_i(t) > \frac{k}{k+1}(t - BD)$.*

Proof. *By definition,*

$$\begin{aligned} \widehat{W}_i(t) = e_i + \sum_{\tau_j \in hp(\tau_i)} \widehat{RBF}_{pj}(\tau_j, t) > t - BD \\ \Rightarrow e_i + \sum_{\tau_j \in hp(\tau_i)} \left(1 + \frac{1}{k}\right) RBF_{pj}(\tau_j, t) \\ \Rightarrow \left(1 + \frac{1}{k}\right) \left(e_i + \sum_{\tau_j \in hp(\tau_i)} RBF_{pj}(\tau_j, t)\right) > t - BD \\ \Rightarrow \left(e_i + \sum_{\tau_j \in hp(\tau_i)} RBF_{pj}(\tau_j, t)\right) > \frac{k}{1+k}(t - BD) \end{aligned} \tag{5.64}$$

Thus proving the theorem.

We are now to prove the theorems bounding the approximation.

Theorem (2 - Bounded Delay). *In a synchronous periodic task set representing a pjd event model and a resource model given by the bounded delay function $f(t) = t - BD$, task τ_i is feasible under a deadline monotonic scheduling policy if:*

$$\exists t \in (0, D_i] : \widehat{W}_i(t) \leq t - BD \tag{5.65}$$

Proof. *Again, let us suppose a $t_0 \in (0, D_i]$ so that $\widehat{W}_i(t) \leq t - BD$, then by lemma 1 - BD $W_i(t) \leq t - BD$. For theorem 1 - BD we have that task τ_i is schedulable under DM.*

Theorem (3 - Bounded Delay). *In a synchronous periodic task set representing a pjd event model and a resource model given by the bounded delay function $f(t) = t - BD$, task τ_i is infeasible under a deadline monotonic scheduling policy for a processor with $(1 - \epsilon)$ of the original capacity, if:*

$$\forall t \in (0, D_i] : \widehat{W}_i(t) > t - BD \tag{5.66}$$

Proof. *By contradiction, let us assume that for all* $t \in (0, dl_i]$ *the approximate cumulative request-bound function lies above* $f(t)$*, i.e.* $\widehat{W}_i(t) > t - BD$*, but the task* τ_i *is still feasible on a* $(1-\epsilon)$ *processor. Since we have* $1 - \frac{k}{k+1} \leq \epsilon$ *and for theorem 1 - BD there is a* $t_0 \in (0, D_i]$*, so that* $W_i(t_0) \leq (1-\epsilon)(t_0 - BD) \leq (\frac{k}{k+1})(t_0 - BD)$*. Nevertheless according to lemma 2 - BD* $\widehat{W}_i(t_0) > t_0 - BD$ *implies* $W_i(t_0) > \frac{k}{k+1}(t_0 - BD)$*, which contradicts the assumption, thus proving the theorem.*

The same proofs can easily be extended to the pjd model, we just have to consider the shifting given by BD in the analysis.

5.6 Outgoing Curves Derivation

At this point, in order for the PJD^* component to be complete, we need to derive the outgoing event curve and the remaining resource curve. While computing the remaining resource curve is not complicated, the outgoing events curve has to be carefully considered, as we will now show.

For the remaining resource curve what we need to do is to add every arrival curve with a simple plus operation and subsequently use a GPC component in order to derive the curve.

On the other hand, the derivation of every outgoing event stream is not straightforward. Due to a deeper analysis needed to study this particular behaviour, this was not considered as part of this thesis and will be mentioned as future work in the concluding chapter.

Chapter 6

Algorithm Implementation and Evaluation

In this chapter we report the implemented algorithm and the evaluation that has been done. For the implementation we chose to use the MATLAB®programming language, in order to have a common interface with the RTC-Toolbox.

A better evaluation of the PJD* component and real-time calculus would be to use its extension, i.e. finitary real-time calculus. However the framework was never openly published. Moreover, the original RTC-Toolbox is based on a closed source Java-Kernel for the curves' representation. A solution to this would have been an own implementation of finitary-RTC, which was however beyond the scope of this thesis.

6.1 Pseudo-Code

The algorithm we implemented takes as input a task set T and of course an approximation factor ϵ. The first part of the algorithm serves as an interface to the whole framework, see algorithm 1. The input tasks are given as parameters to the algorithm with the desired approximation factor.

If the task set is feasible, then the algorithm gives a positive answer and additionally return the approximate worst-case response time of the lowest priority task.

In the algorithm described in 1, after the usual initialisation, the value of k is derived from the approximation factor ϵ (line 2). After that we compute the necessary testing set of the task set (line 3).

The for-cycle in lines 4-6 is used to compute the approximate cumulative request bound function for the $n-1$ higher priority task, with the nested for-cycle computing it for the scheduling points of the set $\widehat{\mathcal{S}}_n$.

Finally, line 8 returns the feasibility test of the computed approximate cumulative bound function, and returna the wcrt, if successful.

Algorithm 1: ApproximationAlgorithm

Data: $e[n], p[n], j[n], d[n], D[n]$: task set
ϵ: approximation factor
Result: feasible: feasibility of the task set
wcrt: worst-case response time

```
1  initialise variables;
2  k = ⌈1/ε⌉ − 1;
3  Ŝ_n = TestingSet (e[n], p[n], j[n], d[n], D[n], k);
4  for i ← 1 to n − 1 do
5      for j ← 1 to SizeOf (Ŝ_n) do
6          Ŵ[j] = Ŵ[j] +ApproxRbf (i,j)
7  for j ← 1 to SizeOf (Ŝ_n) do
8      Ŵ[j] = C[n] + Ŵ[j] ; /* Add the computation time for the last task */
9  feasibile, wcrt = Schedulable (Ŵ[n], S_n);
10 if feasible then
11     return wcrt
12 return feasible;
```

We will now illustrate the algorithm for the computation of the scheduling point set. In this case, we chose to separate the different task models, even though it could be done within the same piece of code. However for clarity and debugging purposes this solution was much more suitable to our needs.

The algorithm is self explanatory, it just implements equations 5.10, 5.20 and 5.34 reported in chapter 5. The only critical point lies within the burst case. In fact, as we have already shown, we need to consider both the burst region and the jitter region. If the number of steps given by k is satisfied within the burst region, then we would have a periodic behaviour given by $d[i]$-times the number of steps.

On the other hand, if the burst region is not sufficient to cover the accuracy parameter, then we need to add the edges of the jitter region, since it would be now dominated by the minimum inter-arrival distance, i.e. the period between two consecutive point is described by the task period $p[i]$. Therefore, we introduced in algorithm 2 the else clause in line 20.

Additionally, as we have seen in the previous chapter, we always need to consider the first job release in the jitter region, even if this would not be covered by the k number of steps, hence line 21.

The algorithm will then return the scheduling set points after sorting the array and eliminating the same values appearing for two or more tasks.

Algorithm 2: TestingSet

Data: $e[n], p[n], j[n], d[n], D[n]$: task set
k: accuracy constant
Result: $\widehat{\mathcal{S}}_n$: array of the scheduling points

```
1  initialise variables;
2  switch TaskModel do
3      case periodic
4          for i ← 1 to n − 1 do
5              for j ← 1 to k − 1 do
6                  AppendTo(Ŝ_n, p[i] ∗ j)
7          AppendTo(D[n])
8      case jitter
9          for i ← 1 to n − 1 do
10             for j ← 1 to k − 1 do
11                 AppendTo(Ŝ_n, p[i] ∗ j − j[i])
12         AppendTo(D[n] − j[n])
13     case burst
14         for i ← 1 to n − 1 do
15             t_burst := (j[i]∗d[i]) / (p[i]−d[i]);
16             for j ← 1 to k − 1 do
17                 if j ≤ ⌈t_burst / d[i]⌉ − 1 then
18                     AppendTo(Ŝ_n, d[i] ∗ j)
19                 else
20                     AppendTo(Ŝ_n, p[i] ∗ j − j[i])
21         AppendTo(t_burst)
22 SortAndPrune(Ŝ_n);
23 return Ŝ_n
```

Moving back to the main interface, algorithm 1 computes the approximate request-bound function for the scheduling points of $\widehat{\mathcal{S}}_n$. The algorithm is again straightforward and implements the equations of the approximate request-bound functions w.r.t. the task model considered.

The periodic and jitter cases are easy to understand, however the burst case needs once more a careful implementation. In fact, in this case, we not only have a simple if-then-else clause, but we need to consider how the burst region and approximation region are related

Algorithm 3: ApproxRBF

Data: i: task, t_j: scheduling point
Data: $\widehat{RBF}(i, t_j)$: approximate request-bound function

```
1  initialise variables;
2  switch TaskModel do
3      case periodic
4          if t_j ≤ (k − 1) ∗ p[i] then
5              R̂BF(i, t_j) = ⌈t_j / p[i]⌉ ∗ e[i]
6          else
7              R̂BF(i, t_j) = (t_j + p[i]) ∗ e[i] / p[i]
8      case jitter
9          if t_j ≤ (k − 1) ∗ p[i] − j[i] then
10             R̂BF(i, t_j) = ⌈(t_j + j[i]) / p[i]⌉ ∗ e[i]
11         else
12             R̂BF(i, t_j) = (t_j + p[i] + j[i]) ∗ e[i] / p[i]
13     case burst
14         if ⌈t_j / d[i]⌉ < ⌈(t_j + j[i]) / p[i]⌉ then
15             if t_j ≤ (k − 1)d[i] then
16                 R̂BF(i, t_j) = ⌈t_j / d[i]⌉ ∗ e[i]
17             else
18                 R̂BF(i, t_j) = (t_j + d[i]) ∗ e[i] / d[i]
19         else
20             if t_j ≤ (k − 1) ∗ p[i] − j[i] then
21                 R̂BF(i, t_j) = ⌈(t_j + j[i]) / p[i]⌉ ∗ e[i]
22             else
23                 R̂BF(i, t_j) = (t_j + p[i] + j[i]) ∗ e[i] / p[i]
24 return R̂BF(i, t_j)
```

to each other. Due to the minimum function used to mathematically describe this model, we always have two different functions possibly representing a time point. Moreover, for a considered time point t, we need to check if the approximating steps are already satisfied or not (lines 15 and 20). Algorithm 3 implements therefore equation 5.25.

What is now left to do is to verify the conditions of schedulability for the task model and, if successful, to derive the worst-case response time respectively. In this case the

Algorithm 4: Schedulability

Data: $\widehat{W}[n]$: approximate cumulative request-bound function
$\widehat{\mathcal{S}}_n$: testing points
Result: feasible: boolean for feasibility
wcrt: approximate worst-case response time if feasible

```
1  initialise variables;
2  if Bounded Delay then
3      Ŝ_n = Max(Ŝ_n − BD, 0)
4  switch TaskModel do
5      case periodic
6          for i ← 1 to SizeOf (Ŝ_n) do
7              if Ŵ[i] ≤ Ŝ_n then
8                  feasible = true;
9                  wcrt = Ŵ[i];
10                 break;
11             else
12                 feasible = false;
13     case jitter
14         for i ← 1 to SizeOf (Ŝ_n) do
15             if Ŵ[i] ≤ Ŝ_n then
16                 feasible = true;
17                 wcrt = Ŵ[i] + j_i;
18                 break;
19             else
20                 feasible = false;
21     case burst
22         /* Implemented Separately                                  */
23         BurstCase (Ŝ_n,)
24 return feasible, wcrt
```

schedulability test depends also on the resource model we implemented, so we need to differentiate the two cases for the full availability and the bounded delay.

This can be done by modifying the time point given by the testing set. In fact, the bounded delay case shifts the line $f(t) = t$ by the value BD, but function has to remain above zero. Our implementation checks whether the resource model depicts a bounded

Algorithm 5: SchedulabilityBurst

Data: $\widehat{RBF}_{pjd}[n]$: approximate cumulative request-bound function

$\widehat{\mathcal{S}}_n$: testing points

Result: feasible: boolean for feasibility

wcrt: approximate worst-case response time if feasible

initialise variables;

`/* Get the index of the last job released */`

I[`SizeOf` $(\widehat{\mathcal{S}}_n)$] = `LastReleased` $(\widehat{\mathcal{S}}_n, \widehat{RBF}_{pjd}[n], e_i)$;

for $i \leftarrow 2$ **to** `SizeOf` $(\widehat{\mathcal{S}}_n)$ **do**

  if *I[i]>lastactive* **then**

    nextjob=lastjob+1;

    y = computation request released by higher priority jobs at time $\widehat{\mathcal{S}}_n(i-1)$;

    wa = y + nextjob*$e[i]$ + $\widehat{RBF}_{pjd}(\widehat{\mathcal{S}}_n(i-1))$;

    wb = nextjob*$e[i]$ + $\widehat{RBF}_{pjd}(\widehat{\mathcal{S}}_n(i))$;

    `/* Check if intersecting */`

    intersection = `GetIntersection` $([\widehat{\mathcal{S}}_n(i-1), \widehat{\mathcal{S}}_n(i)], [wa, wb])$;

    `/* Determine response time */`

    $rt = intersection - (nextjob - 1) * d[n]$;

    `/* Determine worst-case response time */`

    $wcrt = \max(wcrt, rt)$;

    `/* Check if busy period has terminated */`

    wa = y + $I(i)$*e_i + $\widehat{RBF}_{pjd}(\widehat{\mathcal{S}}_n(i-1))$;

    wb = y + $I(i)$*e_i + $\widehat{RBF}_{pjd}(\widehat{\mathcal{S}}_n(i))$;

    intersection = `GetIntersection` $([\widehat{\mathcal{S}}_n(i-1), \widehat{\mathcal{S}}_n(i)], [wa, wb])$;

    $rta = intersection - (I(i) - 1) * d[n]$;

    lastjob=I(i);

    if $rta <= d[n] \&\& \widehat{\mathcal{S}}_n(i) \leq t_{n,burst}$ **then**

      lastactive = -1;

      break;

    if $rta <= p[n] \&\& \widehat{\mathcal{S}}_n(i) > t_{n,burst}$ **then**

      lastactive = -1;

      break;

if *lastactive != -1 ;* `/* Level-i busy period not terminated */`

then

  nextjob=`LastReleased` $(\max(\widehat{\mathcal{S}}_n))$+1;

  wcrt=$\frac{nextjob*e[n]+\sum_{\tau_j \in hp(n)}\left(e[j]+\frac{j[j]}{p[j]}\right)}{1-\sum_{\tau_j \in hp(n)} U[j]} - (nextjob-1)p[n] + j[n]$;

if $wcrt \leq D[n]$ **then**

  feasible = true;

return feasible, *wcrt*

delay, and, if so, subtracts BD to every time point t in the set $\widehat{S}_n$ (see line 2-3 in algorithm 4).

After this, the algorithm proceeds to differentiate the event model considered, however for the periodic and jitter case the procedure remains similar, since we only have to check one job of task τ_i. For every scheduling point the algorithm checks increasingly whether its cumulative request-bound function lies below line $f(t)$, which could either be t or $t - BD$, see lines 7 and 15 in 4.

If such a time point exists, then the algorithm computes the wcrt w.r.t. the event model considered, see lines 9 and 17 in 4.

For the burst case, we need to consider potentially more than one job in the burst region, hence the algorithm not following the same procedure as in the previous cases.

In order to complete the computation of the wcrt, we need to determine whether the level-i busy period has finished or not between two scheduling points. To do that, we need the index of the first job released within two scheduling points. Equation 5.53 provides the index of the last job released, which has been completed by time t. Line 3 of algorithm 5 implements this relation for every scheduling point of $\widehat{\mathcal{S}}_n$.

With line 4 we proceed to check every scheduling point. At first we verify if a job completes within the interval considered $(\widehat{\mathcal{S}}_n(i-1), \widehat{\mathcal{S}}_n(i)]$. If so, we compute the sum of all execution requests released by higher priority jobs at time $\widehat{\mathcal{S}}_n(i-1)$ (Line 7).

After that, lines 8-9 calculate the coordinates of the cumulative request-bound function at points $\widehat{\mathcal{S}}_n(i-1)$ and $\widehat{\mathcal{S}}_n(i)$, in order to derive the response time of the completed job (Line 13). Furthermore, if the level-i busy period terminates within this interval, the intersection point for the cumulative request-bound function and job $I(i)$ has to be found. If so, then the computation stops and the algorithm returns the calculated wcrt. If not, then we proceed to check the other scheduling points.

If after considering all the testing point in the scheduling points set the level-i busy period has not yet terminated, then we can use equation 5.57. Lines 25 to 28 implements this situation.

The algorithm has been modelled over the one reported by Nguyen et al. in [26].

6.2 Evaluation

In this section we present the evaluation results derived through the comparison between the PJD* component and a chain of GPC components of the original RTC-Toolbox.

In order to perform a meaningful comparison between our implementation and the RTC-Toolbox, we had to introduce some limitations for the generated task sets. Firstly, we bound the task periods and jitters. In our task generator we chose a period of at most 10 time units. For longer periods, the RTC-Toolbox is extremely slow: for example, already for task with $p = 20$, on one side the RTC-Toolbox incurs in the `OUT-OF-MEMORY` error

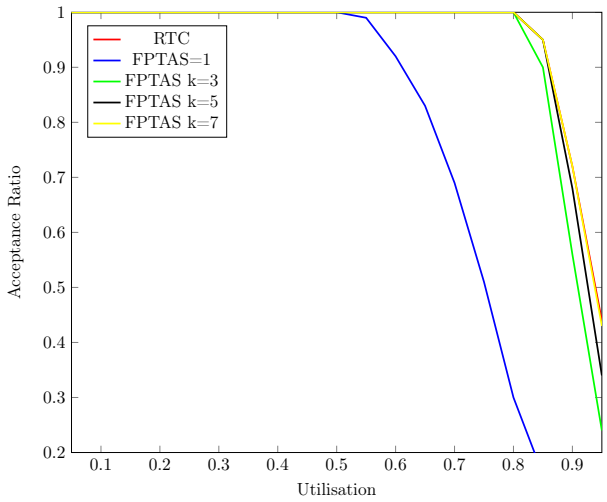

Figure 6.1: Acceptance Ratio. Periodic tasks. RTC and FPTAS

more often, on the other side it takes up to one minute to complete a single experiment over a task set, whereas our framework computes it in $0,014$ seconds.

For all these reasons, the evaluation performed between the two frameworks has to undergo some hard restrictions. Furthermore, alongside the FPTAS, we also have implemented the exact analysis theory, so that we could also perform some evaluation independently from the RTC-Toolbox.

For the experiments we implemented a task generator based on the UUniFast algorithm of Bini et al. [6]. The generator takes as input the total utilisation desired and returns a vector of tasks' utilisation uniformly distributed in $\mathcal{O}(n)$:

```
% UUniFast from Bini and Buttazzo
sumU = utilisation; % the sum of n uniform random variables
utilisations = zeros(1,n_tasks);  % initialization
for i=1:n_tasks-1,
    nextSumU = sumU.*rand^(1/(n_tasks-i)); % the sum of n-i
        uniform random variables
    utilisations(i) = sumU - nextSumU;
    sumU = nextSumU;
end
```

The same MATLAB function `rand` is used to generate random periods and jitters, with periods chosen between $[1, 10]$, and jitters depending on the task model. For the model with constrained jitters, they were strictly smaller than the respective task periods. When considering implicit deadlines, we set $p_i = D_i$. For bounded deadlines we used the `rand` to generate deadline smaller than or equal to the period. For arbitrary deadlines we generated a random value between the interval $[1, 20]$.

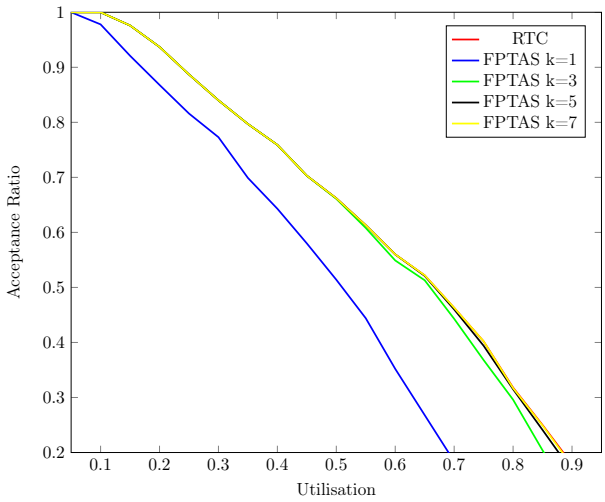

Figure 6.2: Acceptance Ratio. Periodic tasks with jitter. RTC and FPTAS

For the periodic with burst model, the jitter could be at most 20-time units. Moreover, for this model we also used the `rand` function to generate the minimum inter-arrival distance, which is bounded by the period, as in the original RTC-Toolbox.

After the task generation, the PJD* framework is called with the implementation of the above algorithms.

We observed in our experiments, that the computed exact worst-case response time was always the same as the delay calculated by the RTC-Toolbox, while, as expected, the FPTAS returned a wcrt depending on the accuracy factor considered. In figure 6.1 we show the results for periodic tasks and $k = \{1, 3, 5, 7\}$ respectively, note that we cannot distinguish the curves given by RTC-Toolbox and the exact analysis as they are the same. As we can see, the higher the approximation factor, the smaller the difference between the two curves.

All the evaluation results were performed for 100 experiments for every utilisation step, i.e. every utilisation from $0,5$ to $0,95$ with a $0,05$ step. We could not include an utilisation equal to 1 because the RTC-Toolbox, in most cases, does not terminate its computation.

In the following figure 6.2, we considered the periodic tasks with jitter model, again with $k = \{1, 3, 5, 7\}$. As mentioned above, the jitter is bounded by the period. In this case the resulting curves are not monotonically decreasing, since the influence of the jitter can prevent the schedulability of a task set, even if the utilisation is smaller.

For the event model of periodic tasks with burst, we had an unexpected result. While for higher approximation factors, the acceptance ratio remains as expected, for a lower ϵ, our implementation returns a positive feasibility also when the RTC Toolbox does not (see figures 6.3).

Even though our framework carefully implements the model developed in the previous chapters, the results differ from what expected.

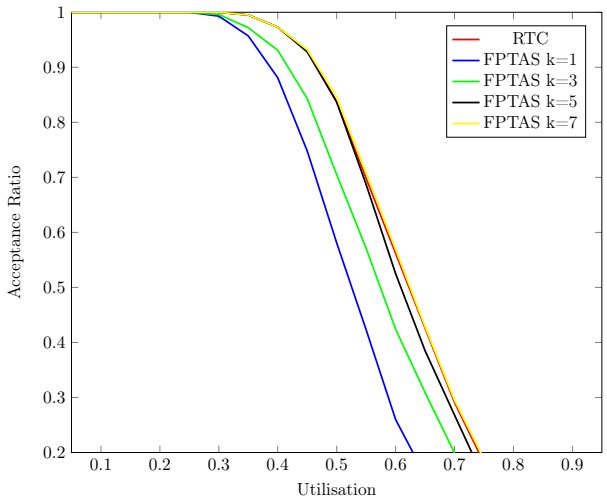

Figure 6.3: Acceptance Ratio. Periodic tasks with burst. RTC and FPTAS

	Periodic	**Periodic with Jitter**	**Periodic with Burst**
RTC	80.3	81.2	123.9
FPTAS	0.5	0.7	0.8

Table 6.1: Run-Time Comparison between RTC and FPTAS for periods $\leq$ 10. Time is in milliseconds.

One problem arises from the fact that for this model, we could not perform an evaluation for a utilisation larger that $0,75$. This was due to the fact that most of the time the computation of the RTC-Toolbox failed. The reason was given by the error RTC-error `X and Y must be wide-sense increasing curves`. As we have seen in chapter 3, the RTC-Toolbox introduces some level of approximation for the represented curves, though this approximation is not bounded.

The arrival curves representing a pjd model are much more complex than the ones for periodic tasks or periodic tasks with jitter. In fact, while the last two models are essentially depicted by a curve with period p after an offset $p - j$, the burst task model has a more expressive aperiodic part, and the combination of other GPC components leads to a not periodical resource curve.

In addition, the problems encountered by using non-integer computation times within the RTC-Toolbox, suggest that an analysis over the curve representation within the Java-Kernel of RTC has to be done in order to better understand where these unexpected results come from.

	Periodic	Periodic with Jitter	Periodic with Burst
RTC	540.3	602.5	772.7
FPTAS	0.5	0.6	0.7

Table 6.2: Run-Time Comparison between RTC and FPTAS for periods $\leq$ 20. Time is in milliseconds.

6.2.1 Run-Time Comparison

To conclude, we present the results obtained analysing the run time of our implementation and the RTC Toolbox. The FPTAS Framework outperforms the original real-time calculus implementation by up to a factor 200 in the burst case (see table 6.1), even for periods smaller than 10.

If we bound the periodic to 20 time units (see table 6.2), then the run-time for real-time calculus drastically increases. For the simple periodic tasks case we have a speed-up ratio of factor 1000, whilst the FPTAS remains on the same scale. Furthermore, we observed in our experiments that whilst the average computation time of real-time calculus could still be acceptable, we also have peaks were the computation hangs for more seconds. For example, in the periodic with burst task model, for periods bigger than 20 time units, the computation time can be up to 7 seconds, whereas the FPTAS remains constant.

The same behaviour was indeed reported in the finitary real-time calculus paper [17]. Moreover, the authors point out that for the same task set, if just one period is increased by two time units, the computation time required by the framework increases exponentially. This due to the period explosion problem reported in chapter 3. In fact, if the periods considered are prime between each other, the time needed to complete the execution drastically increases.

Due to the impossibility of accessing the Java-Kernel of RTC, we had to limit our evaluation to a task set formed by three tasks. In fact, the MATLAB interface of the RTC-Toolbox uses the Java-Kernel to represent the curve. We are not able then to perform some sort of for-cycle to represent the curve: for example, it is not possible to use the task generator we implemented in order to initialise n event curves for RTC, instead this needs to be done by hand. Furthermore, as we started our implementation, the new MATLAB version was not compatible with the RTC-Toolbox [1].

In addition, the RTC-Toolbox fails to cope well with computations times represented by double variables. In fact, it was not possible to perform a reasonable evaluation if the computations were represent with number with four digits decimal number, which is how the double variables are represented in MATLAB. Again, the closed source Java-Kernel

[1] A new released of the RTC-Toolbox was only recently published in mid September 2015, towards the conclusion of this work, see `http://www.mpa.ethz.ch/Rtctoolbox/Overview`, last consulted on 01.11.2015

could not be investigated to understand the reason of this error. The MATLAB interface performs the RTC operation over the curves that are however implemented within the Java Library. Nevertheless we could perform a stable evaluation if the decimals were limit to three digits. Here we report the error we got:

```
Error using rtcfloor (line 23)
Java exception occurred:
java.lang.IllegalArgumentException: CurveSegments in periodicPart
  are not strictly increasing.
```

6.3 Complexity of the FPTAS

As we can see from algorithm 2, the number of points contained in the set $\widehat{\mathcal{S}}_i$ will be at most:

$$1 + (i-1)(k) \tag{6.1}$$

that is, one for the relative deadline of the task, and at most $(k-1)$ steps for every higher priority task plus one testing point for the approximate request-bound function.

The computation of the approximate cumulative request-bound function is also depending on this value. Moreover, the schedulability has to be also evaluated for this number of points.

In our case, we consider only the testing points for the last task, however the framework could be adapted to evaluate the feasibility and *wcrt* for every tasks. Therefore, the complexity would have to take into account at most $\sum_{i=1}^{n} 1 + (i-1)(k-1)$, which gives an overall complexity of $\mathcal{O}(n^2 k)$.

Since parameter k is related to ϵ by $k := \lceil \frac{1}{\epsilon} \rceil - 1$, we therefore have a complexity of $\mathcal{O}(\frac{n^2}{\epsilon})$: this is the complexity given by a fully polynomial-time approximation scheme, as we have seen in chapter 4.

Particular attention should be given to the case of a periodic task with burst. As we have seen before, we would need to potentially check l jobs within the burst region. However, it has been demonstrated that we only have to consider the first and the last jobs released within the interval delimited by two adjacent scheduling points of $\mathcal{S}_i$, and this can be done in constant time $\mathcal{O}(n)$.

For this reasons we can conclude that our approximation scheme for a $PJD*$ is indeed a fully polynomial-time approximation scheme.

Theorem (Complexity of PJD*). *Given a task set T of n tasks $\tau_i : i = 1, ..., n$ with implicit deadline and described by a triple p, j, d. Given an approximation factor $\epsilon \in (0, 1)$, there exists an algorithm A_ϵ, such that it runs in polynomial time both in n and ϵ, i.e. $\mathcal{O}(\frac{n^2}{\epsilon})$. The algorithm has the following properties:*

1. *if the task set is infeasible on a 1-capacity processor, than the algorithm returns the infeasibility*

2. *if the task set is feasible on a (1-ϵ)-capacity processor, than the algorithm returns the task set as being feasible. Furthermore, it derives an approximate worst-case response time bounding from above the exact wcrt. The approximate response time derives is similarly bounded from above by the $\frac{1}{1-\epsilon}$-exact wcrt.*

3. *in other cases the algorithm can either return the task set as being feasible or infeasible.*

Chapter 7

Conclusions

We presented a different approach to analyse particular task models and scheduling policies within the real-time calculus framework. In our work, we introduced a link between classical scheduling theory and real-time calculus.

Due to the generality of real-time calculus, we firstly developed an approximate analysis for periodic task models, and then we moved onto more expressive use cases in order to always cover the most important task patterns originally implemented in RTC.

We have shown that for implicit deadlines, i.e. a rate monotonic scheduling policy, the PJD* component can be easily implemented for a pseudo-polynomial feasibility test within the MPA framework. Moreover, we have shown how a periodic task with burst behaves differently due to the intrinsic arbitrary deadlines derived for the burst region. However the performance analysis developed for this task model is much more complex that the one introduced for periodic tasks and periodic tasks with jitter.

The approach taken throughout this work was to start with the simplest cases and then move to more complex behaviours. Nevertheless the research needed to cover all the possible patterns used in the MPA framework would have needed more than the required six months of work.

In fact, although our implementation was extended in order to also cover the bounded delay case, the time division multiple access resource model needs much deeper analysis. In fact, as we have seen in the introductory chapters, the resource curve of this model is not any more represented by a single line, but has a more complex behaviour instead. On one side, it is possible to extend our implementation without any approximation of the TDMA-resource function. On the other side, an approximation of the behaviour of the same could be introduced. Again, the analysis of this resource model needs further research in the field of TDMA.

We also presented a quick derivation for the remaining resource curve, given by adding all the incoming events and performing a convolution operation with the resource curve.

We eliminated in this way the need of computing a remaining resource curve for every task of the considered set.

Furthermore, we have shown that our implementation can speed up the analysis by an average factor of 200, but it increases with the total utilisation. Moreover, the original RTC-framework does not cope well with task set of more than 4-5 tasks, whereas our implementation of the FPTAS was tested for up to 100 tasks without problems. In addition, in order to perform a comparison between the two frameworks we also needed to limit the periods of the tasks, otherwise the RTC framework failed to finalise the analysis due to the `OUT-OF-MEMORY` error, whilst our implementation did not have such a limitation.

However, our implementation can still be improved by introducing a derivation for the outgoing event curves. In fact, at this point, this represents a huge drawback in our framework, since it does not permit a full integration within the RTC-Toolbox.

Finally, as a future work, we believe that an extensive comparison between our framework and finitary real-time calculus is indeed necessary. In particular, the derivation of the remaining resource curve and every outgoing event curve can benefit from the reduced complexity in finitary real-time calculus.

To conclude, the PJD*-framework we presented still lacks completeness with respect to all the possible resource and event models within real-time calculus. However, it introduces a novel idea to cope with some of the huge drawbacks of real-time calculus. In addition, it would be interesting to adapt the framework to also consider dynamic scheduling policies as earliest-deadline first or some more complex behaviours such as fixed-priority servers or dynamic-priority servers.

List of Figures

List of Algorithms

Bibliography

[1] Albers, K. and F. Slomka: *Efficient feasibility analysis for real-time systems with EDF scheduling.* In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 492–497 Vol. 1, March 2005.

[2] Albers, Karsten: *Approximative Real-Time Analysis, PhD. Thesis.* 2008.

[3] Alur, Rajeev and David L Dill: *A Theory of Timed Automata.* Theoretical Computer Science, 126:183–235, 1994.

[4] Audsley, Nc, Alan Burns, M.F. Richardson and A.J. Wellings: *Hard Real-Time Scheduling: The Deadline-Monotonic Approach.* Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software, pages pp. 133–137, 1991.

[5] Baccelli, Francois, Guy Cohen, Geert Jan Olsder and Jean-Pierre Quadrat: *Synchronization and linearity.* 1992.

[6] Bini, Enrico and Giorgio C. Buttazzo: *Measuring the performance of schedulability tests.* Real-Time Systems, 30(1-2):129–154, 2005.

[7] Boudec, Jean-Yves Le and Patrick Thiran: *NETWORK CALCULUS A Theory of Deterministic Queuing Systems for the Internet.* 2004.

[8] Buttazzo, Giorgio C.: *Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications, Springer.* 2008.

[9] Chakraborty, Samarjit, Simon Künzli, Lothar Thiele, Andreas Herkersdorf and Patricia Sagmeister: *Performance evaluation of network processor architectures: Combining simulation with analytical estimation.* Computer Networks, 41(5):641–665, 2003.

[10] Chen, Jian-jia, Wen-hung Huang and Cong Liu: *k2U: A General Framework from k-Point Effective Schedulability Analysis to Utilization-Based Tests.* page 16, 2015.

[11] Cormen, Thomas, Charles Leiserson, Ronald Riverst and Clifford Stein: *Introduction to algorithms.* The MIT Press, 3rd edition, 2009.

[12] Cruz, Rene L.: *A calculus for network delay. I. Network elements in isolation.* Information Theory, IEEE Transactions on, 1991.

[13] Cuninghame-Greene, Raymond: *Minimax Algebra*, 1979.

[14] Fisher, Nathan and Sanjoy Baruah: *A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines.* Proceedings - Euromicro Conference on Real-Time Systems, 2005:117–126, 2005.

[15] Fisher, Nathan and Sanjoy Baruah: *A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines.* Proceedings - Euromicro Conference on Real-Time Systems, 2005:117–126, 2005.

[16] Fisher, Nathan and Sanjoy Baruah: *Approximation Algorithms for Feasibility Analysis in Real-Time Static-Priority Systems.* pages 1–42, 2005.

[17] Guan, Nan and Wang Yi: *Finitary real-time calculus: Efficient performance analysis of distributed embedded systems.* Proceedings - Real-Time Systems Symposium, pages 330–339, 2013.

[18] Hendriks, Martijn and Marcel Verhoef: *Timed automata based analysis of embedded system architectures.* In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., April 2006.

[19] Künzli, Simon, Arne Hamann, Rolf Ernst and Lothar Thiele: *Combined approach to system level performance analysis of embedded systems.* Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis - CODES+ISSS '07, page 63, 2007.

[20] Lehoczky, John: *Fixed Priority Scheduling of Periodic Task.* Integration The Vlsi Journal, pages 201–209, 1990.

[21] Lehoczky, John, Lui Sha and Ye Ding: *The rate monotonic scheduling algorithm: exact characterization and average case behavior.* Proceedings. Real-Time Systems Symposium, 1989.

[22] Liu, Jyh Charn and James W. Layland: *Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment.* Journal of the Association for Computing Machinery, 20(1):46–61, 1973.

[23] Mok, Aloysius K., Xiang Feng and Deji Chen: *Resource partition for real-time systems.* Proceedings Seventh IEEE Real-Time Technology and Applications Symposium, 2001.

[24] Naedele, Martin, Lothar Thiele and Michael Eisenring: *Characterizing variable task releases and processor capacities.* TIK-Report n45, 1999.

[25] Nguyen, Thi Huyen Chau, Pascal Richard and Nathan Fisher: *The Fully Polynomial-Time Approximation Scheme for Feasibility Analysis in Static-Priority Systems with Arbitrary Relative Deadlines Revisited.* 18th International Conference on Real-Time and Network Systems, pages 21–30, 2010.

[26] Nguyen, Thi Huyen Chau, Pascal Richard and Emmanuel Grolleau: *An FPTAS for Response Time Analysis of Fixed Priority Real-Time Tasks with Resource Augmentation.* IEEE Transactions on Computers, 9340(c):1–1, 2014.

[27] Nguyen, Thi Huyen Chau, N. S. Tran, V.H. Le and Pascal Richard: *Approximation scheme for real-time tasks under fixed-priority scheduling with deferred preemption.* Proceedings of the 21st International conference on Real-Time Networks and Systems - RTNS '13, 2:265, 2013.

[28] Perathoner, Simon: *Modular performance analysis of embedded real-time systems: improving modeling scope and accuracy.* PhD thesis, ETH Zürich, 2011.

[29] Perathoner, Simon, Ernesto Wandeler and Lothar Thiele: *Evaluation and comparison of performance analysis methods for distributed embedded systems.* Master's thesis, Swiss Federal, (276), 2006.

[30] Phan, Linh T X, Samarjit Chakraborty and P. S. Thiagarajan: *A multi-mode real-time calculus.* Proceedings - Real-Time Systems Symposium, pages 59–69, 2008.

[31] Richard, Pascal, Joël Goossens and Nathan Fisher: *Approximate Feasibility Analysis and Response-Time Bounds of Static-Priority Tasks with Release Jitters.* Proceedings of the 15th conference on Real-Time and Network Systems, pages 105–112, 2007.

[32] Richter, Kai: *Compositional scheduling analysis using standard event models.* PhD. Thesis, Electrical Engineering, page 236, 2004.

[33] Richter, Kai, Dirk Ziegenbein, Marek Jersak and Rolf Ernst: *Model composition for scheduling analysis in platform design.* Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324), pages 1–6, 2002.

[34] Santinelli, Luca and Liliana Cucu-Grosjean: *Toward probabilistic real-time calculus.* ACM SIGBED Review, 8(1):54–61, 2011.

[35] Schwiegelshohn, Uwe and Lothar Thiele: *Dynamic min-max problems.* Discrete Event Dynamic Systems: Theory and Applications, 9(2):111–134, 1999.

[36] Stankovic, John A. and Krithi Ramamritham: *What is predictability for real-time systems?* Real-Time Systems, 2(4):247–254, 1990.

[37] Stoimenov, Nikolay: *Compositional design and analysis of distributed, cyclic, and adaptive embedded real-time systems.* (PhD Thesis, 19619), 2011.

[38] Suppiger, Urban, Simon Perathoner, Kai Lampka and Lothar Thiele: *A simple approximation method for reducing the complexity of Modular Performance Analysis.* TIK-Report No. 329, pages 1–10, 2010.

[39] Thiele, Lothar, Samarjit Chakraborty and Martin Naedele: *Real-time calculus for scheduling hard real-time systems.* 2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353), 4(March):101–104, 2000.

[40] Thiele, Lothar and Ernesto Wandeler: *Real-Time Calculus (RTC) Toolbox.*, 2006. Last consulted on 1.11.2015.

[41] Wandeler, Ernesto: *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems, PhD Thesis.* 2006.

[42] Wandeler, Ernesto, Lothar Thiele, Marcel Verhoef and Paul Lieverse: *System architecture evaluation using modular performance analysis: A case study.* International Journal on Software Tools for Technology Transfer, 8(6):649–667, July 2006.

ERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kentlich gemacht habe.

Dortmund, den 2. November 2015

Vasco Fachin