Mälardalen University Press Dissertations
No. 13

Engineering Strength Response-Time Analysis

A Timing Analysis Approach for the Development of Real-Time Systems

Jukka Mäki-Turja

May 2005

Mälardalen Research and Technology Centre (MRTC)
Department of Computer Science and Electronics
Mälardalen University
Västerås, Sweden

Copyright © Jukka Mäki-Turja, 2005
E-mail: jukka.maki-turja@mdh.se
ISSN 1651-4238
ISBN 91-88834-57-3
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

ABSTRACT

When developing computer systems that are part of larger systems, as in control systems for cars, airplanes, or medical equipment, reliability and safety is of major concern. Developers of these systems want to keep the production and development costs to a minimum while maximizing customer benefit by increasing the functionality of the product.

Increasing the number of functions, along with the added complexity that it entails, places a demand on better development methods, models, and tools. Response-Time Analysis (RTA) can be a useful method for these systems by able to guarantee a system's temporal behavior. This thesis presents new techniques aimed at improving currently existing RTA methods. Specifically, these techniques lead to the following improvements:

- The precision in the calculated response times are significantly higher than with previous methods, with typically 15% shorter response times.
- The analysis, itself, can be made more 100 times faster than with previous implementations.

By combining these independent techniques of precise (tight) response times and fast analysis, as shown in this thesis, one can get the benefits of both in a single analysis method. High precision response-time estimates enable either increased functionality within a given hardwaren cost, or a lower cost for a given functionality. Faster RTA will increase the usefulness of RTA by enabling the use of RTA in development tools for real-time systems with hundreds, or even thousands of tasks.

RTA can be particularly beneficial for safety critical applications that have even higher requirements on reliability and safety, and often undergo expensive and lengthy certification processes. We illustrate the possible advantages by applying RTA for tasks with offsets in a real industrial context. The benefits consist of simplifying the development process as well as enabling an efficient resource usage.

SWEDISH SUMMARY

Vid utvecklingen av datorsystem som är en del av en större produkt, som t.ex. styrsystemet i en bil, ett flygplan eller medicinsk utrustning, ställs det ofta mycket hårda krav på säkerhet och tillförlitlighet. En av målsättningarna är även att hålla nere produkt- och utvecklingskostnaden, samtidigt som man vill öka kundnyttan genom att öka innehållet, dvs. funktionaliteten.

Ökningen i antalet funktioner, och komplexiteten den medför, ställer krav på bättre utvecklingsmetoder, -modeller och -verktyg. Att analysera svarstider genom s.k. responstidsanalys (RTA) är ett sätt att kunna garantera systemets tidsbeteende innan produkten tas i drift. I denna avhandling presenteras nya tekniker som syftar till att förbättra existerande RTA-metoder, vilket konkret leder till följande förbättringar:

- Precisionen i de beräknade svarstiderna blir avsevärt högre än tidigare (typiskt ca 15% kortare responstider).
- Analyserna kan göras avsevärt snabbare än tidigare (typiskt ca 100 ggr snabbare än tidigare).

Genom att kombinera dessa två helt oberoende tekniker, behöver man ej offra precision för snabb analys eller vice versa. Det bästa av två världar uppnås i en och samma analysmetod; snabb analystid och precisa svarstider. Högre precision i svarstider möjliggör antingen ökad funktionalitet inom ramen för en given produktkostnad, eller en lägre kostnad för en given funktionalitet. Snabbare analysmetoder innebär att utvecklingsverktyg nu kan använda RTA i praktiken även för riktigt stora system med hundratals, ja även tusentals, funktioner.

Speciellt säkerhetskritiska tillämpningar som måste vara oerhört säkra och tillförlitliga, och många gånger måste genomgå en dyr och tidskrävande certifieringsprocess, kan dra nytta av RTA. En stor del av svensk exportindustri såsom Volvo, Saab, och ABB utvecklar realtidssystem för vilka denna forskning skulle kunna vara av strategisk betydelse.

WHAT IS TIME?
TIME IS WHAT PREVENTS EVERYTHING FROM HAPPENING AT ONCE.

John Archibald Wheeler (The American Journal of Physics, 1978)

PREFACE

The road leading to this Ph.D. thesis has been long and rocky. I have had tons of fun and met people along the way with whom I have shared some wonderful (but sometimes tough) moments. I wouldn't have made it this far without their encouragement, support, and friendship. Therefore, I would like to take this opportunity to express my gratitude to all of them.

First of all, I want to thank my supervisors, Mikael Nolin and Christer Norström, for their invaluable support and friendship throughout the entire process of the development of this thesis. With you, Mikael, I have had endless discussions of the detailed technical aspects as well as the overall big picture (really big after some beers). I must say that I am impressed by how quickly you seem to grasp and find the essence of my sometimes incoherent thoughts. Something I wouldn't have previously believed is that paper writing can actually be fun. However, writing papers with you has actually been a lot fun. You have been a great supervisor. Thanks! And you, Christer, have constantly made an effort to try to broaden my view so I didn't get trapped in the jungle of details and mathematical formulae. Instead, you encouraged me to look at what I have accomplished in a broader context. However, your most valuable asset, as I see it, is that you seem to be able to bring out the best in people, including me. Thanks, Christer, for all your encouragement and support far beyond the call of duty. Thanks also to Hans Hansson and Sasikumar Punnekkat, who by their proof reading, raised the quality of the introductory part of this thesis considerably. Sven Lindow who found errors in papers C and D, which I was able to correct at the last minute, also deserves a thank you.

I would like to thank friends and co-workers at the department who have made the atmosphere both creative and fun. Kaj Hänninen, who recently began the PhD program, has taught me that regardless of how well you might think you understand a problem area, there will always exist questions that have not been addressed before. Don't lose that ability, Kaj. Your curiosity and dedication will bring you and people around you a long way. With Dag Nyström

I have enjoyed some really high moments, sometimes as high as 4000 meters. Now you owe me three times Dag ;-). I really enjoyed our paper writing session at Grundagsätern. It didn't even seem like work, but in paper D you can see the end result. I would also like to thank Sasikumar Punnekkat, who is not only an excellent researcher, but also a very good friend. I appreciate your insightful comments in reviewing papers. Extra thanks also go to Kristian Sandström, Anders Wall, Daniel Sundmark and Åsa Lundkvist for interesting and fruitful research and educational discussions. Ylva Boivie also deserves an extra thank you for her infectious enthusiasm and help in explaining my research to lay people. And of course Harriet Ekwall, who always makes sure my trips are organized in the best way and happily goes that extra mile to make sure you are satisfied, deserves much appreciation.

I would also like to thank some of my best friends who unknowingly have influenced this Ph.D. thesis. First of all, I'd like to thank Sören Helenelund who recommended the bachelor program (no, not the one on TV) to which I got accepted. It turned out pretty well, Sören, don't you think? I have had endless discussions with Joachim Yngvesson about all aspects of life. What our discussions boil down to, I think, is that life prepares you well for the challenges you will be facing on the golf course. I hope we, together with Sören (he introduced me to golf too), will continue to collect empirical evidence for those theories this summer. I have recruited some researchers at the department to strengthen the scientific credibility of our findings. A big thanks also goes to Catherine Reed, my spades buddy in the U.S., who has helped me not only with (well deserved?) breaks from paper and thesis writing but also with help in improving my English. She has also provided me with a glimpse into the field of advanced medical equipment.

Last, but by no means least, I would like to thank my family for never ending love, encouragement and support. My mother, Mirja, has had perseverance to withstand my sometimes stubborn ways. For that, I am truly grateful. My father, Seppo, a craftsman by trade, is a recent and enthusiastic computer user. He cannot, for his life, understand what I have been doing all these years when I cannot even solve his computer problems. In fact, I consider my greatest achievement, related to computers, being able to convince my entire family to acquire them, and to actually use them. Hannu, my younger brother, still believes that "big" in big brother, refers to size. I am sorry to say (or maybe not), you will always be my little brother. Now, you have it in writing ;-).

THANK YOU ALL!

Jukka Mäki-Turja
Västerås a sunny day in April, 2005

LIST OF PUBLICATIONS

Publications included in this thesis

Paper A Jukka Mäki-Turja and Mikael Nolin. *Tighter Response-Times for Tasks with Offsets.* In proceedings of Real-time and Embedded Computing Systems and Applications Conference (RTCSA), Gothenburg Sweden, August 2004.

Paper B Jukka Mäki-Turja and Mikael Nolin. *Efficient Response-Time Analysis for Tasks with Offsets.* In proceedings of the 10th IEEE Real-Time Technology and Applications Symposium (RTAS), Toronto Canada, May 2004.

Paper C Jukka Mäki-Turja and Mikael Nolin. *Fast and Tight Response-Times for Tasks with Offsets.* To appear in proceedings of 17th Euromicro Conference on Real-Time Systems (ECRTS), Palma de Mallorca Spain, July 2005.

Paper D Jukka Mäki-Turja, Kaj Hänninen, and Mikael Nolin. *Efficient Development of Real-Time Systems Using Hybrid Scheduling.* To appear in proceedings of international conference on Embedded Systems and Applications (ESA), Las Vegas USA, June 2005.

I have been the main author and the driving force in developing the ideas presented in these papers which have been supervised by Mikael Nolin. For paper D, Kaj Hänninen provided the basis for the case study that is included in the paper.

Publications not included in this thesis

To present a more comprehensive picture of achievements, during my studies in the area of real-time systems, I list my publications not included in this thesis. See MRTC web[1] for a complete list of also MRTC and technical reports.

Books

- Christer Norström, Kristian Sandström, Jukka Mäki-Turja, Hans Hansson, Henrik Thane, Jan Gustafsson, and Damir Isovic. *Robusta realtidssystem.* MRTC Report ISSN 1404-3041 ISRN MDH-MRTC-25/2000-1-SE, Mälardalen Real-Time research Centre, Mälardalen University, August 2000.

Journals

- Christer Norström, Jukka Mäki-Turja, Jan Gustafsson, Kristian Sandström, and Ellus Brorson. *An Overview of RTT: A Design Framework for Real-Time Systems.* In Journal of Parallel and Distributed Computing, August 1996.

- Jan Gustafsson, Jukka Mäki-Turja, and Ellus Brorson. *Benefits of Type Inference for an Object-Oriented Real-Time Language.* In OOPS Messenger, 7(1), January 1996.

Conferences and workshops

- Kaj Hänninen, Jukka Mäki-Turja, Mikael Nolin. *Industrial Requirements in Development of Embedded Real-Time Systems – Interviews with Senior Designers*. To appear in WiP Session of 17th Euromicro Conference on Real-Time Systems (ECRTS), Palma de Mallorca Spain, July 2005.

- Kaj Hänninen, John Lundbäck, Kurt-Lennart Lundbäck, Jukka Mäki-Turja, and Mikael Nolin. *Efficient Event-Triggered Tasks in an RTOS*. To appear in proceedings of international conference on Embedded Systems and Applications (ESA), Las Vegas USA, June 2005.

[1]http://www.mrtc.mdh.se/publications

- Jukka Mäki-Turja and Mikael Nolin. *Faster Response Time Analysis of Tasks With Offsets*, WiP Session of Real-Time Systems Symposium (RTSS), Cancun Mexico, December 2003.
- Christer Norström, Mikael Gustafsson, Kristian Sandström, Jukka Mäki-Turja, and Nils-Erik Bånkestad. *Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry.* In proceedings of the 8th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, Washington US, April 2001.
- Anders Wall, Kristian Sandström, Jukka Mäki-Turja, and Christer Norström. *Verifying Temporal Constraints on Data in Multi-Rate Transactions.* In proceedings of Real-time and Embedded Computing Systems and Applications Conference (RTCSA), Korea, December 2000.
- Christer Norström, Kristian Sandström, Jukka Mäki-Turja, and Nils-Erik Bånkestad. *Findings from introducing state-of-the-art real-time techniques in vehicle industry*. In industrial session of the 12th Euromicro Conference on Real-Time Systems, Stockholm Sweden, June 2000.
- Jukka Mäki-Turja, Gerhard Fohler, and Kristian Sandström. *Towards Efficient Analysis of Interrupts in Real-Time Systems.* In 11th EUROMICRO Conference on Real-Time Systems, York England, May 1999.

Technical and MRTC reports

- Kaj Hänninen and Jukka Mäki-Turja. *Component technology in Resource Constrained Embedded Real-Time Systems* Technical Report, Mälardalen Research and Technology Centre, March 2004.
- Jukka Mäki-Turja and Mikael Nolin, *Combining Dynamic and Static Scheduling in Hard Real-Time Systems*. MRTC report 71, Mälardalen Research and Technology Centre, Mälardalen University, October 2002.

Other post graduate theses

- Jukka Mäki-Turja. *Smalltalk - a suitable Real-Time Language?* Licentiate thesis no. 639, ISSN 0280-7971, Department of Computer and Information Science, Linköping University, October 1997.

CONTENTS

I

Thesis

CHAPTER 1
Introduction

Modern society is getting more and more dependent on computers and software. This is strikingly illustrated by the following quote:

> "Modern society runs on oil and software, and we think we know how to replace the oil".[1]

When talking about computers and software, most people think of the boxes and screens at our desks, desktop computers. However, the vast majority, more than 99.8% of all computers sold 2002, were embedded in other products [Tur02], such as mobile phones, microwave ovens, cars, or airplanes, etc. In fact, almost every electronic device today contains at least one simple computer. As an example, modern cars – such as the BMW 7-series – contain more than 65 microprocessors [Tur02].

Since computers and software are replacing more and more of the traditional mechanical solutions, the software systems are becoming larger in size. Furthermore, since software is more flexible, the possibility for new types of functionality emerges when switching to software solutions. One example is the anti skid functionality in a modern car. The possibility for this functionality arose from the fact that when switching from mechanically controlled brakes to computer controlled brakes, the wheels can be individually controlled.

This increase in size and diversity of functionality, [HMTN05], presents an increasing challenge for the developers of this kind of systems. One of the main challenges comes from the fact that when replacing mechanical subsystems with software controlled solutions, the computer based systems must be at least as reliable and safe as the mechanical solutions they replace. Consider, e.g., the

[1]Australia's competitive dependence on software by Australian Software Quality Research Institute, 1992, URL:http://www.sqi.gu.edu.au/docs/sqi/misc/Aus_Comp_Depend_onS_W.pdf

control systems for a nuclear power plant, an airplane or medical equipment. We would not accept that these system malfunction or have glitches in the same manner as, e.g., desktop computers. It is not uncommon that desktop computers, every now and then, crash or slow down and have to be restarted.

The increased demand on reliability and safety of control systems means that the functionality of these systems must be verified before the system is actually taken into use.

Another important and complicating factor for this type of system, compared to a desktop system, is the timing (temporal) behavior of the computer system. In the desktop domain, the computer sets the pace of interaction, and its environment (most often human) has to adapt to that pace. For control systems where the computer is controlling a physical device, the roles are reversed. The environment (e.g., car, airplane, robot, power plant, or medical equipment), i.e., laws of nature, dictates the pace of interaction. Therefore, correct timing behavior is a vital part of the overall correctness criteria for these types of computer systems. For this reason, they are called real-time systems.

1.1 Problem formulation

In order to meet the challenges of reliability and safety requirements during the ever increasing complexity of real-time functionality, developers need to have proper development and analysis tools at their disposal.

One of the most important aspects of real-time systems is that they must exhibit a predictable timing behavior. Furthermore, these systems are also often embedded in larger systems where resources are often scarce. Therefore, models and analysis tools aiming at predicting the temporal behavior in a resource constrained environment, are of great concern. One such method is the Response-Time Analysis (RTA) method. By predicting response times, RTA aims at predicting the systems worst case timing behavior.

This thesis investigates to what extent RTA can be extended to fit an industrial development context of predictable embedded real-time systems. In particular, we will:

1. Investigate how RTA could calculate more accurate, i.e., less pessimistic, response times and which would reduce the resource consumption for these systems.

2. Investigate how to make the current RTA analysis methods more efficient with respect to analysis speed. The goal is to achieve an efficient

RTA able to handle large enough task sets for handling real industrial applications.

3. Furthermore, we investigate how a task model with offsets and the corresponding RTA could improve the design process in the development of embedded real-time systems.

1.2 Thesis outline

This thesis is a so called "*sammanläggningsavhandling*"[2] where the main part constitutes of a number of previously published research papers. In addition to the appended papers, a "sammanläggningsavhandling" contains an introductory part consisting of introduction and background to the research area where the research papers make scientific contributions. It also summarizes these contributions. This thesis is thus organized into two parts:

Part I	Contains the introductory part for the thesis which aims at giving an introduction and background to the area of research for the appended papers. Furthermore, this part will also present some related work and discuss aspects of the research presented in this thesis from an industrial viewpoint.
Part II	Contains four appended scientific papers A, B, C and D. These papers appear as published, except for paper C which is an extended version of the published paper. They have only been modified to fit the layout[3] of this thesis.

Part I is organized as follows:

- Chapter 1 has given a short introduction to real-time systems and how such systems differ from other (desktop) systems. It has also motivated temporal analysis of such systems and presents this thesis' problem formulation.

- Chapter 2 gives a more formal definition of a real-time system and defines important terms in the area of temporal analysis. This chapter also presents the field of real-time scheduling. In particular, schedulability analysis is presented which will lead to the area of Response-Time Analysis (RTA).

[2]A doctoral thesis consisting of a collection of previously published research papers.
[3]Discovered typos have also been corrected.

- RTA is the specific research area of this thesis and will be discussed extensively in Chapter 3. Basic RTA, introduced by Joseph and Pandya, will be presented. Furthermore, the work leading to RTA for tasks with offsets will be outlined.
- This thesis' scientific contributions, presented in papers A, B, C and D, together with their impact, are summarized in Chapter 4.
- Chapter 5 presents and discusses some related work in the area of RTA for tasks with offsets.
- Finally, Chapter 6 concludes this thesis and describes possible future work and relevant open problems.

CHAPTER 2
Real-time systems

The fundamental view of a real-time system is that it interacts with, i.e., observes and controls, its physical environment. The main objective is to act and react upon changes in the controlled process. To be able to control a physical process that obeys the laws of nature, the computer control system must be fast enough in order to establish an internal view of the controlled process' state and to be able to produce a relevant response to changes (events) in the process. As an example, consider a computer controlled airbag functionality in the control system for a car, depicted in figure 2.1.

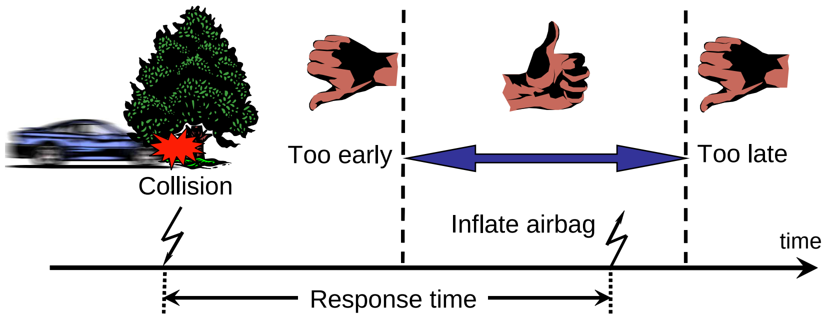


Figure 2.1: Fundamental view of a real-time system

The control system must be able to:

1. Detect that an event (a collision) has occurred.

2. Decide on what action to perform by executing some computations. In this example it could be to determine whether or not the force of impact is severe enough to launch the airbag.

3. Produce a response to the event, i.e., launch the airbag in response to the collision.

The correctness of the airbag functionality is not only dependent on if the computations (deciding whether to launch the airbag or not) are correct, it is also dependent on timely response to the event, i.e., when the control systems acts (launches the airbag). This timing, i.e., the time from the event occurs until the computer system produces a response is called a **response time**, for that functionality.

For the airbag functionality, there is a window of time, see figure 2.1, when the airbag must be launched. If the airbag is launched before that window of time, it will be collapsing by the time the driver hits it, making it useless. The consequences of launching the airbag after the specified window of time, are more serious. If the driver is just about to hit the steering wheel when the airbag inflates, it will explode in his face. This example also shows that real-time systems are not the same as fast systems, rather they must be able to predictably adapt themselves to the pace of their environment. Various definitions of real-time systems exist. The one provided by *The Oxford dictionary of computing* offers one that agrees well with our view as well as being descriptive:

DEFINITION 2.1 (REAL-TIME SYSTEM) *A system in which the time at which the output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag (delay) from the input time to output time must be sufficiently small for acceptable timeliness.*

In a computer based system, there are often many different types of functionality that compete for the computer's resources, such as computing power, memory, and I/O. In the control system for a car there is much more functionality than controlling the airbag. Examples include anti-locking breaking systems (ABS), engine control, cruise control, infotainment systems, and so on. In desktop computers there are lot of different programs that are active at the same, such as, e-mail client, web browser, word processor, MP3 player, etc.

There are two fundamental differences between real-time and desktop applications:

1. A real-time system must be able to handle **all** the critical functionality in a timely fashion, whereas desktop computers strive for maximum **average** throughput. That is, some functionality may suffer, but that is acceptable as long as the average responsiveness is high. In real-time computing, the average response time is of little or no concern.This fact is strikingly illustrated in the analogy by J.A. Stankovic: "*There was a man who drowned crossing a stream with an average depth of six inches*."

2. In a desktop application, the computer dictates the pace of interaction. Most often, this suffices since its environment, probably a human, reacts much slower than the computer. But sometimes, when the computer has many things to do, the user may get annoyed at waiting for a response from the computer. In real-time systems, on the other hand, the environment dictates the pace of interaction and the computer has to keep up and respond to changes in the environment in a timely fashion. Otherwise, the system may not perform correctly, and in the worst case there can be threat to human lives.

Thus, we see that the notion of time and timely response in real-time systems is important. Safety critical systems, where timing faults could mean threat to human lives, place an even greater concern on timeliness. In safety critical systems, correct timing behavior must be guaranteed before the product is deployed. This is often done by lengthy and costly certification processes.

2.1 Types of real-time functionality

Since more and more functionality is being controlled by real-time software, the diversity of this functionality continues to increase. A typical real-time system contains functionality ranging from control (e.g., continuous engine control) to user interaction or diagnostics functionality. Real-time systems can be classified in several ways. One common classification is to distinguish between hard and soft real-time systems:

Hard Failing to meet timing requirements will result in failure of the system. Systems where failures potentially result in catastrophic human consequences are also called safety critical systems.

Soft The system does not fail if a timing requirement is not met, rather the quality of service deteriorates by a late response.

In reality it is unusual that a system contains solely hard or soft functionality. Rather, the functionality is mixed, and/or fits on a grey scale between purely soft and purely hard functionality. For example, in a control system for a car, functionality such as ABS and engine control are considered hard. Functionality such as electronic windows, seat heaters, and infotainment systems can be considered soft. This thesis focuses on analyzing and verifying temporal behavior for hard real-time systems. However, knowledge of timing behavior is also useful in quantifying the quality of service for soft real-time systems.

Another common classification of systems is to distinguish between *time-triggered* (TT) and *event-triggered* (ET) systems. Typically, control functionality is by its nature time-triggered, i.e., the activation of the functionality is controlled by the progress of time. Examples include controlling the water level in a tank or medical equipment monitoring the vital signs, such as a cardiopulmonary monitor[1]. In these instances, the water level and vital signs are checked periodically independently of what happens in the environment. Functionality characterized by sudden changes in the environment is termed event-triggered. Examples of such functionality include user interaction such as detecting an activation of an emergency button. The airbag functionality depicted in figure 2.1 is also a typical example of an event-triggered functionality.

Control systems are generally embedded in larger applications, so called embedded systems. These systems are characterized by having a fixed and limited amount of resources. They are fixed in the sense that when applications are shipped, upgrades are difficult and costly; and they are limited in the sense that the amount of memory and computing power is small. The motivation for manufacturers to limit resources is increased revenue. As an example, consider a modern car which consists of several computers (CPUs), more than 65 for the BMW 7-series. Since the product volumes are high (hundreds of thousands), even small savings in product cost, e.g., hardware resources, will result in a substantial increased revenue.

In conclusion, we see that there is a wide variety of functionality in a real-time system. All this functionality, at least the hard and safety critical parts, have to be guaranteed to meet their timing requirements. Furthermore, for embedded systems, this has to be achieved in a system with limited resources. In order to meet this challenge, developers of real-time systems need proper method, analysis, and tool-support.

[1]Monitors and records blood pressure, pulse rate and rhythm, respirations, and body temperature

2.2 Real-time scheduling

Oxford English Dictionary Online[2] defines scheduling as:

> The action of entering in or drawing up a schedule; esp. the preparation of a timetable for the completion of the various stages of a complex project; the co-ordination of many related actions or tasks into a single time-sequence.

Scheduling in the area of computer systems, according to Encyclopedia Britannica Online[3], is:

> The allocation of system resources to various tasks, known as job scheduling, is a major assignment of the operating system. The system maintains prioritized queues of jobs waiting for CPU time and must decide which job to take from which queue and how much time to allocate to it, so that all jobs are completed in a fair and timely manner.

We see that scheduling, in general, means to decide what to do at each specific point in time, and that scheduling from an operating systems view is to decide what task (job) to assign the CPU. Real-time scheduling aims at ensuring that all tasks in the system will meet their timing requirements under resource constraints.

2.2.1 Task model

The schedulable entity, representing a single thread of control in real time systems is called a *task*. A task is, from a scheduling point of view, an abstraction of functionality and corresponds to a piece of sequentially executable code. A task describes the temporal requirements and constraints of the corresponding functionality by a number of attributes. Common examples of task attributes include:

- **Period**, T_i, specifies how often a task i gets activated, and is inversely proportional to frequency. However, not all tasks are periodic, as we shall later see.

[2]http://www.oed.com/
[3]http://www.eb.com/

- **Worst case execution time**, C_i, specifies the absolute longest time it takes to execute the code of task i, if it could run on the CPU without interruption. There has a been numerous research results covering the topic of establishing a task's WCET. Ermedahl and Engblom *et al.* give good overviews of some of these methods [Erm03, EEN$^+$03].

- **Deadline**, D_i, specifies a constraint on the completion time of task i. The task must finish no later than D_i time units after it has been activated.

A task is said to arrive to the system upon activation. For event-triggered functionality the corresponding task (or tasks) handling that functionality is activated when the triggering event happens. Time-triggered tasks are activated at their scheduled time. Different activation patterns commonly considered in the real-time scheduling community are (see figure 2.2):

- **Periodic** tasks are activated at perfect periodicity, i.e., they are activated at times $0, T, 2T, 3T$, etc.

- **Aperiodic** tasks can be activated at any time and with any frequency, i.e., there is no information about their activations at design time, and thus a hard real-time system is unattainable. However, there exists methods that allow aperiodic tasks in hard real-time systems. These are based on providing either firm guarantees[4] or best effort service[5] for them [But97]. Systems that provide firm guarantees for aperiodic tasks are also known as admission control systems. Examples can be found in .

- **Sporadic** tasks is another approach to deal with aperiodic task activations. Sporadic tasks have an uncertainty on when they are activated. However, they are characterized by having a *minimum inter-arrival time* between two consecutive activations. while there is no information on exactly when task activations will occur their frequency is bounded by the minimum inter-arrival time. This gives that sporadic tasks will have a periodic worst-case activation-pattern with the minimum inter-arrival time as the period. So, from worst case scheduling point of view, sporadic tasks can be treated as if they were periodic.

The collection of tasks that constitute a system is called a *task set*. A task set with corresponding attributes, and the constraints and rules under which

[4]Upon arrival, the task is either admitted to the system and guaranteed to make its deadline or it is rejected.

[5]Task are admitted to the system but no grantees can be given.

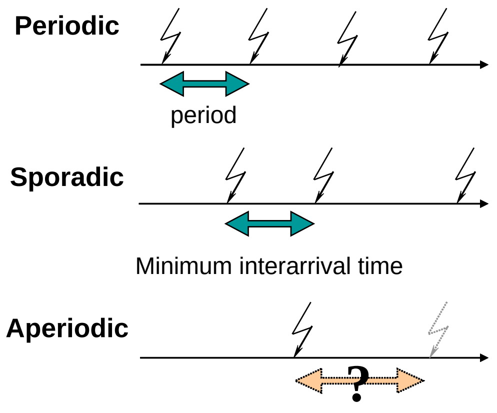


Figure 2.2: Different task activation patterns

tasks execute, is called a *task model*. The task model should ideally reflect as much as possible the requirements of the functionality and the capabilities and constraints of the underlying run-time system.

2.2.2 Scheduling algorithms

A *scheduling algorithm* for embedded real-time systems aims at satisfying the timing requirements of the entire system functionality, i.e., meet all tasks deadline constraints, while minimizing the use of resources. There exist a wide variety of scheduling algorithms in the real-time research literature. These can be classified in many ways: priority-based, value-based, rate-based, server algorithms, etc. One common and coarse grained classification is based on when the actual scheduling decision, i.e., the decision of what task to execute at each point in time, is made. This classification categorizes scheduling algorithms into static and dynamic scheduling:

- **Static scheduling**. The scheduling decisions are made off-line, i.e., before run-time. These decisions are stored in a static schedule. During run-time, the dispatcher simply dispatches tasks according to the predefined schedule. Static scheduling is also commonly referred to as off-line or time-triggered scheduling.
- **Dynamic scheduling**. Scheduling decisions are made on-line by a run-time scheduler. Typically some task attribute, such as deadline or pri-

ority, is used by the scheduler to decide what task to execute. Dynamic scheduling is also commonly referred to as on-line or event-triggered scheduling.

For static schedulers, where every scheduling decision is made at design time, the run-time dispatcher becomes very simple, it just activates tasks according to the predefined schedule. The static scheduler however, is often complex in the sense that it deals with task models with a high degree of expressiveness where tasks have many attributes and complicated constraints.

Dynamic schedulers are often much simpler, both with respect to the expressiveness of the corresponding task model and sophistication of the scheduling algorithm. This is due to the fact that the scheduling decisions are made at run-time and a too sophisticated and powerful algorithm would steal processing power (CPU time) from tasks, resulting in a too large administrative overhead. The most widespread, both in research and in commercial real-time operating systems, is the *fixed priority scheduling* (FPS) algorithm. All major open standards on real-time computing support fixed-priority scheduling [SAr$^+$04].

The pre-run-time configuration activities of fixed priority schedulers consist of assigning priorities to tasks. At run-time, tasks that are activated, either by passing of time or external events, are placed in a ready queue. The dispatcher chooses, each time it is invoked (typically at each clock tick and system call that releases a task), to execute the task with the highest priority among those in the ready queue. In fixed priority scheduling, a task placed in the ready queue is considered *released for execution* (which usually happens at its activation, also known as the task's arrival time).

Further discussions on static versus dynamic (FPS) schedulers can be found in [Loc92, XP00].

2.2.3 Schedulability analysis

A task set is said to be schedulable if a schedule can be found which guarantees that all tasks will meet their timing constraints under all circumstances. Schedulability analysis aims, before run-time, to determine whether a task set is schedulable or not. For most real-time scheduling algorithms some kind of schedulability analysis test is available.

In static scheduling, the schedulability analysis is combined with the construction of the schedule, a so called proof by construction approach. That is, if a schedule which fulfills all timing requirements and constraints can be constructed, the system is, by definition, schedulable.

Real-time research on schedulability in fixed priority scheduled systems has resulted in a wide variety of research results. Several different schedulability-analysis techniques for fixed priority systems exist. The most powerful approach, that provides the highest obtainable utilization and is able handle the most expressive task models, is to use *response-time analysis* (RTA).

CHAPTER 3
Response-time analysis

Liu and Layland [LL73] provide the theoretical foundation for analysis of fixed priority scheduled systems. They define an instant in time, called a **critical instant**, which, if a task is released at that time, will lead to its worst case (longest) response time. For a simple task model with independent periodic tasks the critical instant is defined as:

THEOREM 3.1 *A critical instant for any task occurs whenever the task is released simultaneously with the release of all higher priority tasks.*

PROOF REFERENCE. *The theorem is proved in [LL73].*

For the simple task model used by Liu and Layland this means that the critical instant for the entire system will occur when all tasks are released simultaneously. Liu and Layland also present a utilization based schedulability test under the assumption of independent tasks that have deadlines equal to their period ($D_i = T_i$). Furthermore, priorities must be assigned according to rate monotonic (RM) priority ordering, stating that the shorter the period of a task, the higher the priority. The schedulability test for n number of tasks in the system is then as follows:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

Stating that the total utilization (achieved by summing up all task utilizations C_i/T_i) must be lower than or equal to the expression on the right hand side. This schedulability bound approaches about 69% ($\ln 2$), when n approaches infinity. This schedulability test, although very simple, provides an only *sufficient* condition under which the task set is schedulable. That is, while guaranteeing schedulability upon passing the test, the task set may still be schedulable upon failing the test.

- A schedulability test providing only a *sufficient* condition means that if a task set passes the test, it guarantees that all deadlines will be met under any circumstances. However, if the task set does not pass the test, the task set might be deemed schedulable with some other test. Putting it another way, a sufficient condition might place unnecessary hard restrictions to be able to guarantee schedulability.

- A schedulability test providing only a *necessary* condition means that the task set can not be schedulable upon failing the schedulability test. However, passing the test does not guarantee schedulability. Putting it another way, a necessary condition is insufficient to be able to guarantee schedulability. A trivial example of a necessary condition is that the total utilization must not exceed 100%.

A schedulability test providing both *necessary and sufficient* conditions means that upon passing the test the task set is schedulable and upon failing the test the task set is unschedulable. The necessary and sufficient condition also means that the schedulability test is *exact*. Such a schedulability test, for the simple Liu and Layland task model, is provided by the **response-time analysis** (**RTA**) method. RTA consist of calculating response times for all task and comparing them with corresponding task's deadlines. Tests that merely provide a sufficient condition are also denoted as *approximate* tests, since, e.g., overestimating worst case response times will never wrongfully deem a task set as schedulable.

3.1 Basic RTA

Joseph and Pandya presented the first basic RTA for the simple Liu and Layland task model [JP86]. The task model for basic RTA is as follows. A task τ_i is specified by:

- A period, T_i, specifies either the period of a periodic task or the minimum inter-arrival time of a sporadic task.

- Worst case execution time, C_i, specifies the longest time it takes to execute the code of the task if it could run on the CPU uninterruptedly.

- Deadline, D_i, specifies a constraint on the completion time of the task. The task must finish no later than D_i time units after it has been activated.
- Priority, P_i. A user defined integer value. As opposed to rate monotonic schedulability analysis, priorities can be set arbitrarily.

In addition, the following assumptions must hold in order for the analysis to be valid:

- Tasks must be independent, i.e., there can be no synchronization between tasks.
- Tasks must not suspend themselves.
- Deadlines must be less or equal to corresponding periods, i.e., $D_i \leq T_i$.
- Tasks must have unique priorities.

The following formula determines the worst case response time, R_i, of task τ_i:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ is the set of all higher priority tasks to τ_i.

The smallest positive value which satisfies the above equation corresponds to the worst case response time. Since R_i occurs both at the left and right side of the equation, it cannot be solved directly. Also, R_i cannot be factored out. However, the equation can be numerically solved by the following recurrence relation:

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

which can iteratively be solved using fix-point iteration [ABRW91, ABT$^+$93]. Starting with $R_i^0 = C_i$ and iterating until $R_i^{n+1} = R_i^n$ is guaranteed to yield the smallest possible solution and thus the response time for τ_i [SH98].[1]

EXAMPLE 3.1 (RESPONSE-TIME CALCULATIONS) *As an example, consider the task set in table 3.1 on the following page where the priorities H, M, and L denotes high, medium, and low respectively.*

[1]In order to guarantee convergence either 1) one must ensure a total task utilization is not greater than 100% or 2) one can stop iterating when $R_i^n > D_i$, i.e., a deadline violation has occurred.

Task	T_i	C_i	D_i	P_i
τ_1	4	1	4	H
τ_2	6	2	6	M
τ_3	10	3	10	L

Table 3.1: Example task set

Below we show the fix-point iterations for calculating the response time of the lowest priority task τ_3. In the formulae we omit the subscript 3, so R^k actually denotes R_3^k (the kth iteration in the fix-point calculations when calculating the response time of task τ_3). Figure 3.1 depicts these calculations graphically. At the top of the figure, activation patterns for the two higher priority tasks, τ_1 and τ_2, are depicted. The progress of the fix-point iterations is illustrated beneath, starting with $R^0 = C_3 = 3$. Intuitively, R^1 is obtained by accounting for interference from higher priority tasks during R^0. The figure indicates that both τ_1 and τ_2 are activated one time each during that time, interfering with 1 and 2 units of time, respectively. Hence $R^1 = 6$. This intuition applies to each iteration step until $R^4 = 10$ (higher priority task interference during R^3 is 3 and 4 time units, respectively). Higher priority task interference during R^4 is still 3 and 4 time units and $R^5 = 10$. Thus, a fix-point has been reached and the response time for τ_3, $R_3 = R^5 = 10$, has been obtained.

$$R^0 = C_3 = 3$$

$$R^1 = C_3 + \left\lceil \frac{R^0}{T_1} \right\rceil C_1 + \left\lceil \frac{R^0}{T_2} \right\rceil C_2 = 3 + 1 + 2 = 6$$

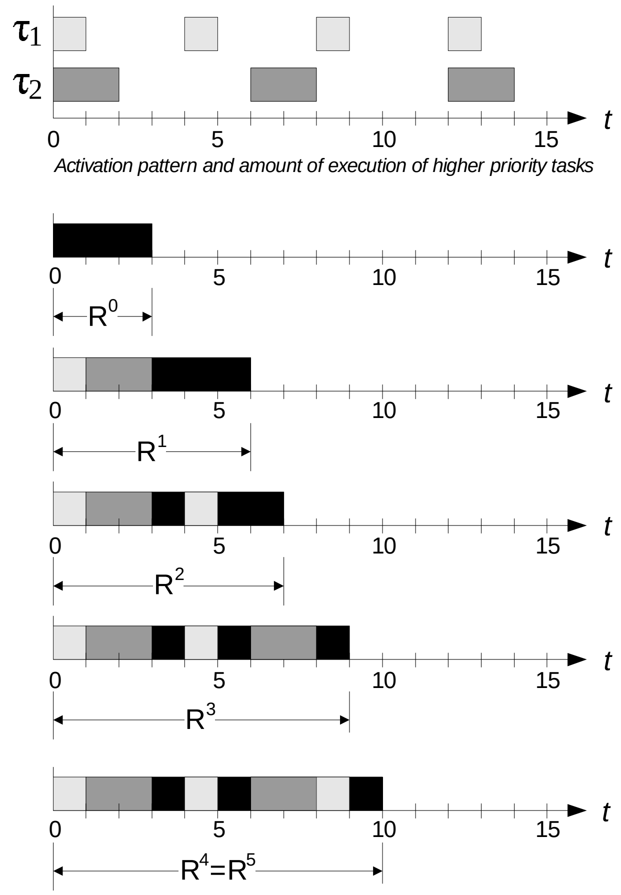
$$R^2 = C_3 + \left\lceil \frac{R^1}{T_1} \right\rceil C_1 + \left\lceil \frac{R^1}{T_2} \right\rceil C_2 = 3 + 2 + 2 = 7$$

$$R^3 = C_3 + \left\lceil \frac{R^2}{T_1} \right\rceil C_1 + \left\lceil \frac{R^2}{T_2} \right\rceil C_2 = 3 + 2 + 4 = 9$$

$$R^4 = C_3 + \left\lceil \frac{R^3}{T_1} \right\rceil C_1 + \left\lceil \frac{R^3}{T_2} \right\rceil C_2 = 3 + 3 + 4 = 10$$

$$R^5 = C_3 + \left\lceil \frac{R^4}{T_1} \right\rceil C_1 + \left\lceil \frac{R^4}{T_2} \right\rceil C_2 = 3 + 3 + 4 = 10$$

$$R^5 = R^4 \;\; \textit{Fix-point has been reached!}$$



τ_1
τ_2
t
0
5
10
15
Activation pattern and amount of execution of higher priority tasks
R^0
R^1
R^2
R^3
R^4=R^5

Figure 3.1: Fix-point iteration example

Since $R_3 \leq D_3$, and assuming τ_1 and τ_2 also meet their deadlines, the task set is schedulable, i.e., all tasks will, under all run-time circumstances, meet their deadlines.

One can view the fix-point and RTA equations as: when the total task execution demand meets the supply of the CPU, all tasks have finished their execution, and thus a fix-point has been reached. Figure 3.2 depicts this viewpoint for the example above. The execution demand of tasks τ_1, τ_2 and τ_3 are highlighted separately and the fix-point iterations steps are illustrated by the dotted line, with each R^k marked with a black dot.

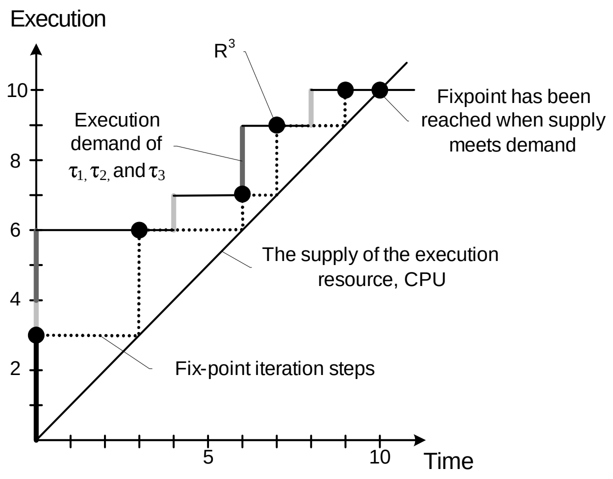


Figure 3.2: Illustration of fix-point iteration steps

3.2 Extending and applying RTA

There have been numerous extensions made to this basic response-time analysis. These extensions aim at either lifting some of the restrictions made in the assumptions in section 3.1 or extending the capabilities of the task model. Some of the extensions improving the applicability of RTA are:[2]

- **Lifting the assumption of independent tasks**. Enabling task communication via shared (locked) resources, such as semaphores, will have the

[2]The RTA formulae for some of these extensions can be found in Appendix A.

effect, on a task's execution, of potential interference also from lower priority tasks. Sha *et al.* [SRL87, SRL90] introduced priority inheritance protocols that solve the priority inversion problem where a task may indefinitely be waiting for a lower priority task to finish. Furthermore, they also presented the priority ceiling protocol (PCP) which prevents deadlocks. An even more important property of PCP is that the blocking time caused by lower priority tasks can be bounded and thus incorporated in the RTA method. Algorithms to calculate blocking times for different resource locking protocols can be found in [But97].

- **Arbitrary deadlines**. Lehoczky lifts the restriction of deadlines being less than or equal to periods [Leh90]. Consequently, several instances of a task can be simultaneously active. The impact of this is that RTA, for a task τ_i, first has to determine the length of a level-i busy period (processor is busy executing tasks with priority higher or equal to τ_i). During that busy period τ_i may have been activated several times, and the worst case response of τ_i is the maximum of all of those corresponding instance response times. Lehoczky provides two sufficient utilization based schedulability tests, while Tindell extends the RTA formulae and thus provides both a sufficient and necessary test in [Tin92a].

- **Accounting for jitter**. This is an extension of the basic task model by considering variations in task periodicity, so called jitter. This kind of variation could occur, e.g., due to precedence constraints, i.e., the activation of a task is dependent on the completion of another. It could also depend on the arrival of a message if analyzing a distributed system. Depending on the variation in execution time (or message sending time in case of a message) of the predecessor, the activation of the successor may vary, even if the predecessor is activated periodically. The deviation from perfect periodicity (can also be viewed as the difference between the earliest and the latest possible release of a task or message) is called *release jitter*. Rajkumar first reasoned about jitter in [RSL88], and it became more explicitly addressed by Tindell and Audsley *et al.* [Tin92a, ABT$^+$93].

- **Analyzing other devices**. RTA was originally designed for analyzing the CPU. However, the basic RTA has been extended to handle other devices such as communication and hard disk devices. Basic RTA assumes a preemptive task model, i.e., when a higher priority task is released, it preempts (interrupts) the execution of the currently running (lower priority) task. When modeling, e.g., communication devices such as CAN and ATM networks, this assumption becomes invalid. A message which has been granted the device has access to the device until completion. The Controller Area network (CAN) can be seen as a non-preemptive fixed priority schedulable resource. In [THW94] Tindell, Hansson, and Wellings extend RTA to analyze message passing in CAN. Ermedahl, Hannson, and Sjödin (now Nolin) shows how RTA can be applied to analyze traffic in an ATM network [EHS97, Sjö00]. Tindell and Burns also provides a method to calculate response times for hard real-time multimedia disks [TB94].

- **Fault tolerant systems**. S. Punnekkat extends the worst-case response time analysis to include fault-tolerant real-time tasks under various fault tolerant strategies with the assumption of a known minimum inter-arrival time between faults [Pun97].

- **Distributed systems**. In [TC94] Tindell and Clark apply RTA in a distributed context where the communication channel is also scheduled and analyzed by RTA. With these results, end-to-end response times can be calculated for tasks scheduled in a distributed system. Precedence relations among tasks (and messages) are modelled by release jitter, i.e., variation in the response time of a predecessor become release jitter for the successor.

- **Modeling OS overhead**. In order for RTA to be used in engineering practice it has to be able to efficiently model real world situations. One such reality is to model operating system overhead. Two examples of taking scheduler overhead into account is Katcher *et al.* [KAS93] and Burns *et al.* [BTW95].

- **Temporal dependencies – Introducing offsets**. For all above methods it is assumed that tasks are arbitrarily phased, which means that the critical instant assumption of simultaneous release of all tasks is in fact possible at run-time. In systems where tasks are temporally dependent, and thus can not be activated simultaneously, the critical instant assumption of [LL73] becomes pessimistic. Tindell introduced the task model with offsets [Tin92b] with a corresponding RTA. Palencia Gutiérrez and González Harbour formalized and extended Tindells work in [PG98] by allowing unlimited release jitter and by introducing dynamic offsets.

A more detailed discussion of some of these improvements can be found in A Practitioners Handbook for Real-Time Analysis [KRP$^+$99]. This book is focused on a practitioner's point of view and thus aims at applying RTA in an engineering context. A historical perspective of real-time scheduling research, where RTA is a big part, can be found in [SAr$^+$04]. This thesis focuses on the task model with offsets and improves upon the corresponding approximate RTA.

3.3 RTA for tasks with offsets

Basic RTA relies on the critical instant assumption that all tasks are released simultaneously. This becomes a pessimistic assumption, i.e., results in unnecessary long response-time estimates if task activations are temporally dependent. This section will give an overview of RTA for tasks with offset as presented in [PG98].

3.3.1 Task model

A system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic (or sporadic) sequence of events with period (minimum inter-arrival time) T_i. The activating events are considered mutually independent, i.e., phasing between them is arbitrary. A transaction Γ_i contains $|\Gamma_i|$ number of tasks, and each task is activated when a relative time denoted as *offset*, elapses after the arrival of the external event. Offset is used to express the temporal dependency between releases of tasks.

A task is denoted by τ_{ij}, where the first subscript, i, denotes which transaction the task belongs to, and the second subscript, j, denotes the number of the task within the transaction. A task is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The system model is formally expressed as follows:

$$\begin{aligned}
\Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\} \\
\Gamma_i &:= \langle \{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i \rangle \\
\tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} \rangle
\end{aligned}$$

There are no restrictions placed on offset, deadline or jitter, e.g., they can each be smaller or greater than the period.

Parameters for an example transaction (Γ_i) with two tasks (τ_{i1}, τ_{i2}) are depicted in Figure 3.3. The offset denotes the earliest release time of a task relative to the start of its transaction, and jitter (illustrated by a shaded region) denotes the variability in the release of the task, i.e., the actual task release can occur anywhere in the shaded region. The upward arrows denote earliest possible release of a task, and the size of the arrow corresponds to the released task's worst case execution time.

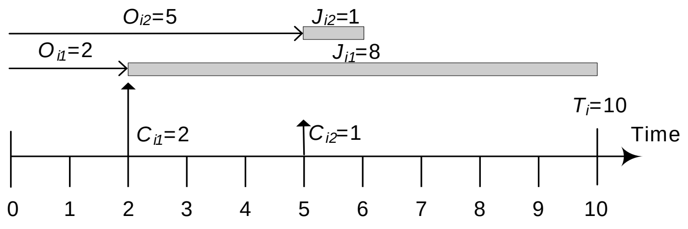


Figure 3.3: Example transaction

In [PG98] dynamic offsets are introduced, i.e., offsets may vary between different activations of a task. However, dynamic offsets are not part of the task model. Rather, they are modelled by static offset and jitter. The definition of a dynamic offset, O_{ij}^d, is that it may dynamically vary between a minimum and a maximum value:

$$O_{ij}^d \in [O_{ij}^{min}, O_{ij}^{max}]$$

This dynamic offset is modelled by the static offset (O_{ij}) and jitter (J_{ij}), producing a new offset (O'_{ij}) and jitter (J'_{ij}) term, as follows:

$$\begin{aligned} O'_{ij} &= O_{ij}^{min} \\ J'_{ij} &= J_{ij} + O_{ij}^{max} - O_{ij}^{min} \end{aligned}$$

Dynamic offsets are often dependent on response times of previous tasks in the transaction. Response times, in turn, are dependent on these dynamic offsets. The solution to this problem is similar to that of calculating response times for tasks with jitter in a distributed system, i.e., starting with response times as zero, and iterating until a stable solution is achieved [TC94].

3.3.2 Exact analysis

Let τ_{ua} denote the *task under analysis*[3]. In the classical RTA (without offsets) the *critical instant* for τ_{ua} is when it is released at the same time as all higher priority tasks. In a task model with offsets this assumption yields pessimistic response times since some tasks can not be released simultaneously due to offset relations. Therefore, Tindell relaxed the notion of critical instant to be:

> At least one task in every transaction is to be released at the critical instant. (Only tasks with priority higher to τ_{ua} are considered.) [Tin92b]

Since it is not known which task coincides with (is released at) the critical instant, every higher priority task in a transaction must be treated as a *candidate* to coincide with the critical instant. Every possible combination of critical instant candidates must be considered. Let $N_i(\tau_{ua})$ denote the number of tasks in transaction Γ_i having priority higher than τ_{ua}.

The number of possible critical instant combinations to examine, when calculating the response time for τ_{ua}, is:

$$\Big(N_u(\tau_{ua}) + 1\Big) \prod_{\forall i \neq u} N_i(\tau_{ua})$$

That is, all possible combinations of higher priority tasks including τ_{ua},[4] must be considered.

Since task attributes such as deadline and jitter are allowed to be larger than periods, several instances of a task may be active simultaneously. Thus, previously activated instances can interfere with the execution of a subsequent instance. So, RTA must take this into consideration. The intuition behind the RTA formulae (for details see paper A, B, or C and Appendix A of the corresponding papers) is as follows:

- For each critical instant combination, assuming we are interested in the response time of τ_{ua}, do:

  1. Calculate the worst case busy period where the processor is busy executing tasks with higher or equal priority tasks than τ_{ua}.
  2. In that busy period a number of instances of τ_{ua} are released. Calculate the response time for each such instance.

[3]This can be read as *task under analysis* as well as task a belonging to transaction Γ_u.

[4]Hence the $+1$ for Γ_u, the transaction of τ_{ua}.

- The worst case response time for τ_{ua} is obtained by selecting the maximum of all such calculated instance response times.

This however, becomes computationally intractable (represents an NP-complete algorithm which grows exponentially with the number of tasks [PG98]) for anything but small task sets. Hence [Tin92b] introduces an approximate RTA algorithm with polynomial complexity.

3.3.3 Approximate RTA

The combinatorial explosion in cases to consider, makes the exact analysis intractable. The heart of the approximate method, introduced by Tindell [Tin92b], is to reduce the number of cases to explore.

During RTA, when calculating a transaction's interference on τ_{ua}, the exact analysis relies on information about which task in each transaction that coincides with the critical instant. Since this is not known, the exact analysis tries every possible combination globally in the system. The approximate algorithm, however, approximates transaction interference by only considering each task within its corresponding transaction as coinciding with the critical instant. The one resulting in the highest interference, at each point in time, approximates the transaction's interference. That is, the approximated transaction interference is the maximum of all variants at each point in time. By restricting its view locally within a transaction, the number of cases [Tin92b] has to consider becomes:

$$\Big(N_u(\tau_{ua}) + 1\Big) + \sum_{\forall i \neq u} N_i(\tau_{ua})$$

We get an additive instead of a multiplicative effect in the number of cases to consider. In order to reduce some pessimism, [PG98] chooses not to use this approximated transaction interference for Γ_u (the transaction τ_{ua} belongs to). Instead, they use the exact method by testing every possibility for Γ_u. The number of critical instant combinations to consider then becomes:

$$\Big(N_u(\tau_{ua}) + 1\Big) * \sum_{\forall i \neq u} N_i(\tau_{ua})$$

This is the variant represented in Appendix A of papers A, B, and C which also includes further discussions and complete formulae.

CHAPTER 4
Thesis contributions

This thesis presents scientific contributions in the area of response-time analysis (RTA) for tasks with offsets, presented by Tindell [Tin92b] and Palencia Gutiérrez and González Harbour [PG98]. This chapter summarizes and discusses the impact of these contributions.

4.1 Problem formulation restated

In order to meet the challenges of reliability and safety requirements in face of the ever increasing complexity of real-time functionality, developers need to have adequate development and analysis tools at their disposal.

One of the most important aspects of real-time systems is that they must exhibit a predictable timing behavior. Furthermore, these systems are also often embedded in larger systems where resources often are scarce. Therefore, models and analysis tools aiming at predicting the temporal behavior in a resource constrained environment, are of great concern. One such method is the Response-Time Analysis (RTA) method. By analyzing response times, RTA aims at predicting the systems worst case timing behavior.

This thesis investigates to what extent RTA can be extended to fit an industrial development context of predictable embedded real-time systems. In particular, we will:

1. Investigate how RTA could calculate more accurate, i.e., less pessimistic, response times which would reduce the resource consumption for these systems.

2. Investigate how to make the current RTA analysis methods more efficient with respect to analysis speed. The goal is to achieve an efficient RTA

which is able to handle large enough task sets for handling real industrial applications.

3. Furthermore, we investigate how a task model with offsets and the corresponding RTA could improve the design process in the development of embedded real-time systems.

4.2 Summary of contributions

The contributions of this thesis, in the context of RTA for tasks with offsets, consist of:

(1) A Technique that enables tighter approximate response-time estimates.

(2) A Technique that enables considerably faster analysis of approximate response times.

(3) A Technique that enables the combination of (1) and (2) in one single analysis method.

(4) Quantitative evaluations of the above techniques.

(5) Showing how RTA for tasks with offsets can simplify the design tradeoff between static and dynamic scheduling.

These contributions are presented in papers A, B, C, and D, which are enclosed in part II of this thesis. These papers are:

Paper A Jukka Mäki-Turja and Mikael Nolin. *Tighter Response-Times for Tasks with Offsets.* In proceedings of Real-time and Embedded Computing Systems and Applications Conference (RTCSA), Gothenburg Sweden, August 2004.

Paper B Jukka Mäki-Turja and Mikael Nolin. *Efficient Response-Time Analysis for Tasks with Offsets.* In proceedings of the 10th IEEE Real-Time Technology and Applications Symposium (RTAS), Toronto Canada, May 2004.

Paper C Jukka Mäki-Turja and Mikael Nolin. *Fast and Tight Response-Times for Tasks with Offsets.* To appear in proceedings of 17th EUROMICRO Conference on Real-Time Systems (ECRTS), Palma de Mallorca Spain, July 2005.

Paper D Jukka Mäki-Turja, Kaj Hänninen, and Mikael Nolin. *Efficient Development of Real-Time Systems Using Hybrid Scheduling.* To appear in proceedings of international conference on Embedded Systems and Applications (ESA), Las Vegas USA, June 2005.

I have been the main author and the driving force in developing the ideas presented in these papers which have been supervised by Mikael Nolin. For paper D, Kaj Hänninen provided the basis for the case study that is included in the paper.

In sections 4.3 to 4.6 the contribution of each paper is further detailed, followed by description of the collected contribution and impact of this thesis in section 4.7.

4.3 Paper A

Jukka Mäki-Turja and Mikael Nolin. *Tighter Response-Times for Tasks with Offsets.* In proceedings of Real-time and Embedded Computing Systems and Applications Conference, Gothenburg Sweden, August 2004.

Paper A shows how existing approximate RTA for tasks with offsets [Tin92b, PG98] result in unnecessary pessimistic response times. We reveal and exploit a misconception, resulting in an overestimation of higher priority task interference in the response-time formulae, dating back to the original response-time analysis presented by Jospeh and Pandya [JP86]. This misconception has gone undetected because overestimation in higher priority task interference does not produce any pessimism in response times for the classical model or for the exact analysis for task with offsets. This is due to the fact that fix-point convergence cannot be reached during time intervals where this overestimation occurs. However, using approximate RTA for tasks with offsets, this overestimation produces overly pessimistic response times. This comes from the fact that the overestimated interference-function is not used directly in the fix-point iterations, instead they are first subjected to a maximization function. This situation can be compared to arithmetic calculation with rounded up floating point values (integers); the most accurate result would be obtained by using floating point values, and only at the end do the round up, instead of rounding up every value before each calculation step.

In the paper we redefine the interference function, reduce the pessimism, and show how the corresponding response time formulae change. Formal proofs show that our modified RTA never yield response times that are longer than those obtained with [Tin92b, PG98], and that it still provides safe, i.e., never underestimates, response times.

Using this new interference function, imposed interference as we call it, significantly shorter response times can be obtained. Simulation results show that typically 15% tighter (i.e. shorter) response time can be obtained compared to [Tin92b, PG98]. In certain situations the response time can be improved upon (tightened) by more than 50%. In the special case with only one high priority transaction, corresponding to, e.g., a static schedule (see paper D), our approximate method yields exact response times.

On the downside, our algorithm has to pay a penalty of slower fix-point convergence, and thus longer analysis time. Since it models higher priority task interference more accurately, it needs more iteration steps to reach a fix-point.

4.4 Paper B

Jukka Mäki-Turja and Mikael Nolin. *Efficient Response-Time Analysis for Tasks with Offsets.* In proceedings of the 10th IEEE Real-Time Technology and Applications Symposium, Toronto Canada, May 2004.

In paper B we address the analysis time of the approximate RTA for tasks with offsets. We present a technique which enables an efficient implementation of the RTA method presented in [PG98]. The resulting method calculates response-times in a considerably shorter analysis time.

The main effort in performing RTA for tasks with offsets is to calculate how higher priority tasks interfere with the task under analysis. The nature of this interference is that it exhibits a repetitive periodic pattern for each transaction. Furthermore, within that period it increases at discrete points in time. The essence of our method is to calculate and store this information (i.e., the discrete points for the duration of one period) statically and during fix-point calculations, use a simple table lookup. We formally prove that the RTA-equations can be reformulated to allow such a static representation of task interference.

Simulations show that the speedup for our method compared to [PG98] is substantial. For realistically sized task sets (100 tasks), performing schedulability analysis for an entire task set gives speedups of about 50 times. Since we have reduced the complexity the relative improvement will be even higher for larger task sets, which means that arbitrary large speedups can be achieved by just scaling up the task set. In an on-line RTA context, e.g., on-line admission control, where interference from tasks in the same transaction as the task under analysis can be ignored, our method outperforms previous methods by a factor of more than 100 and reduces the actual time from the millisecond to the microsecond range, for the task sets considered.

4.5 Paper C

Jukka Mäki-Turja and Mikael Nolin. *Fast and Tight Response-Times for Tasks with Offsets.* To appear in proceedings of 17th EUROMICRO Conference on Real-Time Systems, Palma de Mallorca Spain, July 2005.

In this paper, we bring together the two independent improvements of paper A and paper B. The method in paper B for fast analysis is not directly applicable for the tight analysis of paper A. The enabling factor for a static representation of the original approximate RTA [PG98] is that the higher priority task interference has a periodic pattern and that it increases at discrete points in time. Our tight method, presented in paper A, defines a new interference function which does not increase at discrete points in time and thus their approximate interference will not exhibit a simple periodic pattern.

In paper C, we find a repetitive pattern also for the approximate imposed interference of paper A. We show how this continuous interference function can be represented and approximated by a discrete interference function without losing any precision in resulting response-time estimates. Proofs are provided to show that all of our manipulations are safe and produce the same response times as those in paper A.

In a simulation study we obtain speedups of more than two orders of magnitude for realistically sized tasks sets compared to the tight analysis (essentially the same result as comparing the original approximate RTA in [PG98] with our fast approach in paper B). We also demonstrate that the fast-and-tight analysis has comparable execution time to that of the fast analysis. We see that introducing pessimism in modeling the interference function in points where a fix-point can not be reached, we completely eliminate the penalty introduced by our tight analysis presented in paper A. Hence, we conclude that the fast-and-tight analysis is the preferred analysis method when fast analysis and tight response-times estimates are needed. We do not need to sacrifice accuracy for speed, both are obtained with the fast-and-tight analysis.

4.6 Paper D

Jukka Mäki-Turja, Kaj Hänninen, and Mikael Nolin. *Efficient Development of Real-Time Systems Using Hybrid Scheduling.* To appear in proceedings of The 2005 International Conference on Embedded Systems and Applications, Las Vegas USA, June 2005.

This paper is an engineering paper showing how RTA for tasks with offset can be used in an industrial setting.

When developing real time systems, there is a design trade off in choosing scheduling strategy. Two of the most commonly used scheduling strategies are the static off-line scheduling and the dynamic fixed priority scheduling. Since both scheduling strategies have their pros and cons, a hybrid static and dynamic scheduling model would simplify the design tradeoff of which scheduling strategy to choose. Choosing the most appropriate strategy for each function, instead of force-fitting it to an overall strategy for the entire system, will simplify this trade off. Furthermore, such a hybrid system not only simplifies the design choices but also gives the possibility to save system resources and improve responsiveness.

Even though there have been successful attempts in incorporating both strategies in the same system, most provide only best effort service to dynamic tasks [Arc, Flx]. In this paper we show how RTA for tasks with offset is able to model such a hybrid system and provide tight response-time guarantees also for dynamic tasks.

An industrial case study at Volvo Construction Equipment [Vol] using the commercial real-time operating system Rubus by Arcticus [Arc], demonstrates how this approach enables more efficient use of computational resources, resulting in a cheaper or more competitive product since more functionality can be fitted into legacy, resource constrained, hardware.

4.7 Impact of contributions

This section outlines potential impact of the contributions presented in this thesis.

4.7.1 Impact of fast and tight RTA

A tight RTA has the benefit of not overestimating resources. In the case of RTA, the resources consist of processing (CPU) power. While not actually reducing the system utilization at run-time, RTA is able to guarantee schedulability for task sets with higher utilization since the overestimation in response times is reduced. The exact analysis for tasks with offsets is, of course, the tightest possible solution to use. However, it is computationally intractable for anything but small task sets. Thus, the approximate RTA is a good trade-off, and the tight RTA presented in this thesis produces the tightest response times of all comparable approximate methods available.

RTA is not only used as a schedulability analysis tool, but it is also used in a wider context. For example, schedulability analysis is performed in the inner loop of optimization or search techniques such as task attribute assignment and control performance enhancement.

Many task attribute assignment techniques have a schedulability test by RTA to test if a certain attribute configuration yields a schedulable system. For example, Gutiérrez García and González Harbour present a method for priority assignment for tasks and messages in [GG95]. Bate and Burns also present a method to assign priorities, offsets, and deadlines where each configuration is tested by a schedulability test [BB99]. Sandström and Norström (former Eriksson) present a genetic algorithm approach where the algorithm assigns task offsets and priorities in order to fulfil original more complex constraints [SN02].

Cervin recognizes in [Cer99] that in achieving high control performance, one needs to incorporate scheduling information in the control design. He proposes an attribute (deadlines and priorities) assignment algorithm for enhancing control performance. Also, this algorithm relies on a schedulability test to ensure that a control enhancement step does not violate schedulability.

Since RTA is performed several times to test if a configuration is schedulable, task attribute assignment and control performance enhancement methods require the implementation of RTA to be efficient in order to be useful in engineering tools. One can deduce that since RTA can be used for realistically sized task sets with hundreds or even thousands of tasks, also these methods could be applied to larger class of systems using such an RTA method. These methods would, most probably even to a higher degree than RTA itself, benefit from a fast RTA since the it is performed repeatedly for many different configurations. The number of configurations to test can be several thousands for solving a problem such as task allocation.

With the combined fast and tight RTA method presented in this thesis, the traditional trade-off situation between a fast RTA on one hand, and accurate response times on the other, is eliminated. This new *tight and fast* method exhibits the most accurate response times as well as the fastest analysis time among all comparable approximate RTA methods.

4.7.2 Impact of RTA in an engineering context

The task model with offsets simplifies the design trade-off between static and dynamic scheduling by enabling a choice between them on a per function level instead of system level. An offset relation represents a temporal dependency between the activation of two tasks. Offsets can be viewed as a modeling concept, i.e., it is used to model situations where tasks have temporal dependencies among them, in order for RTA to produce more accurate response times. Examples of offsets used as a modeling concept contains self suspending tasks and precedence constraints.

With the task attributes, jitter and offsets, one can model tasks that suspend themselves. Palencia Gutiérrez and González Harbour show in [PG98] how a task which suspends itself can be modelled by two separate tasks. Assume that a task suspends itself for S time units. This task would be modelled as one transaction (Γ_i) consisting of two tasks τ_{i1}, corresponding to code before suspension, and τ_{i2}, corresponding to code after suspension. The activation time of τ_{i2} depends on the completion time of τ_{i1} and the suspension time S. Thus offset and jitter for τ_{i2} will be: $O_{i2} = S + R_{i1}^{BC}$ and $J_{i2} = R_{i1}^{WC} - R_{i1}^{BC}$ (where R^{BC} denotes best case response time, for which it is always safe to assume zero, and R^{WC} denotes worst case response time).

We see that for self suspending tasks, the latter task's activation is dependant on the former task's completion. This kind of effect also appears in analyzing distributed systems where tasks and messages have precedence relations [PG98]. A task which depends on the completion of a preceding task or the arrival of a message will be assigned an offset. The offset, in this case represents the minimum delay relative to the event that triggered the transaction, the task will suffer before it will be released for execution. The jitter, on the other hand, represents the longest possible delay before the task is released.

Offsets can also be viewed as a design concept where they can be used to achieve some other goal. However, in order to use offsets as a design parameter, the run-time system has to be able to enforce these offsets. Paper D shows such a situation where a static scheduler activates time-triggered tasks, thus enforcing offsets. Consider a real-time system containing control functionality, with control performance as the main goal, coexisting with event-triggered functionality where high responsiveness is the main goal. Some capabilities of offsets as a design concept for such systems could include:

- **More complex constraints can be faithfully modeled**. Time-triggered control functionality can be scheduled by a static scheduler and assigned offsets that the run-time system can enforce.

  The main advantage of this approach is that it can use a powerful off-line scheduler with an expressive task model which can handle more complicated task constraints. For example, in many control systems, jitter is detrimental on control performance [Cer99, NGS$^+$01]. A static scheduler could have, in addition to schedulability, a goal to enhance control performance by minimizing jitter. It can do this by assigning offsets for sampling and actuating tasks in the system.

- **A higher degree of responsiveness can be obtained**. By not only having a priority to determine the degree of responsiveness, but also an additional offset attribute, there exist better means to assign responsiveness where it is needed the most. Assume an approach, proposed by Cervin and Bate *et al.* [Cer99, BNC03], where the short sampling and actuating part of the control functionality are modelled as separate tasks with high priorities and offsets in order to minimize jitter. The actual control calculations in between could be given lower priority as long as it finishes before actuation starts. The updating of the control state could, with the same argument, be given even a lower priority as long as it finishes before next sampling. Then responsiveness of tasks with priority lying in between sampling and control calculation (or between actuation and updating control state) is enhanced.

  Furthermore, the static scheduler can also be given an objective to produce a schedule that enhances the responsiveness for tasks with lower priority than those in the schedule by preventing long busy periods. The highest degree of responsiveness for low priority tasks will be when the static schedule contains equidistant gaps in the entire schedule.

- **Eliminating the need for synchronization protocols**. Another advantage of being able to separate tasks in time, is that tasks accessing shared resources can use time separation by assigning offsets instead of using expensive synchronization protocols.

The introduction offset concept in FPS systems brings possibilities that were previously exclusive for static scheduling. Being able to choose between static and dynamic scheduling on a per function basis instead of an overall system level, simplifies the design trade-off between static and dynamic scheduling. This will enable developers to use more suitable strategies for different functionalities in a system.

CHAPTER 5
Related work on tasks with offsets

This section describes some related work in the area of RTA for tasks with offsets, and their relation to the work of this thesis.

5.1 Variant of exact RTA

Redell presents a variant of the exact worst case RTA for tasks with offsets and release jitter in [RT02]. All tasks in the system may have mutual offset relations, not only tasks belonging to the same transaction. The approach is to unfold the schedule for the duration of the least common multiple (LCM) of all task periods and consider each task instance separately. The worst case response time for a task is obtained by selecting the task instance that results in the longest response time. The main advantage of the method is its time complexity over schedule simulation method presented in [Aud91]. However, it still suffers from the complexity problems that all exact RTA methods for offsets do. That is, having to investigate numerous possible critical instant scenarios. Choosing the task periods unwisely, i.e., relative prime periods, will result in an even worse explosion in cases to consider.

5.2 Best case RTA

In systems where release jitter contributes to the resulting response times, as for example, when analyzing end-to-end response times in a distributed system [TC94, PG98], it is desirable to keep jitter as low as possible. In [PGG98] Palencia Gutiérrez and González Harbour tighten the worst case response times by calculating also a lower bound on the best case response time for all tasks.

Precedence-related tasks in the system are modelled by offsets and release jitter. A task is assigned an offset which corresponds to its predecessors best case response time, representing the earliest possible release of the task. The worst case response time of the predecessor represent the latest possible release of the task. Thus, the difference between earliest and latest release of a task will be the succeeding tasks release jitter. By not assuming an indefinitely small best case response time for tasks, the release jitter is minimized. Hence, the worst case response time becomes less pessimistic. In [RS02], Redell and Sanfridsson provide an exact variant by identifying a favorable instant, corresponding to the critical instant for the worst case RTA. The favorable instant will lead to the shortest possible response time for a task. Another benefit of minimizing jitter, is that it increases RTAs applicability for control systems where small jitter enhances control performance [Cer99, NGS$^+$01, BNC03].

5.3 Improving RTA using precedence information

In systems where tasks have precedence constraints such as distributed systems, RTA methods such as [TC94, PG98] can yield pessimistic results, i.e., unnecessarily long response times. During RTA, many different critical instant task combinations are considered. Each combination results in a specific mutual phasing between tasks. To obtain the response time, the combination resulting in the longest response time is chosen. However, with precedence relations some of these combinations may become impossible. Consider a task chain (related by precedence) of three tasks with priorities high, low, and high respectively. Assume further a task of middle priority for which we are interested to calculate the response time. In this situation, the middle priority task cannot experience interference from both higher priority tasks since they have a low priority task between them. Using such precedence information, many cases that can not occur at run-time can be ignored during RTA. Thus, the pessimism in resulting response times can be reduced. Palencia and Harbour introduced this concept for linear transactions, i.e., where each task can have one successor [PG99]. Redell extended this approach to allow a task to have several successors, so called tree-shaped transactions in [Red04].

5.4 RTA for earliest deadline first systems

Palencia Gutiérrez and González Harbour provides an RTA method for tasks with offsets scheduled under earliest deadline first (EDF) scheduling policy

[PG03]. The method highly resembles the fixed priority counterpart, and since the exact analysis is computationally intractable they provide a similar approximate technique. However, the analysis time of the approximate technique is significantly higher than for the fixed priority case. Since both offset based methods, the one for fixed priority and the one for EDF, can be combined in analyzing distributed systems, heterogeneous systems consisting of some EDF nodes and some FPS nodes can feasibly be analyzed for end-to-end response times.

5.5 Relation to work presented in this thesis

This thesis contributes to the theory of RTA for tasks with offsets in fixed priority systems in two ways. First, by introducing the concept of imposed interference, which more accurately captures the nature of higher priority task interference, the approximate RTA is able to produce less pessimistic response times. Second, this thesis recognizes that higher priority task interference exhibits a periodic pattern which can be stored statically, and during fix-point iteration perform a simple table lookup. This will result in considerably faster approximate RTA. The work of this thesis relates to the work presented in this chapter as follows:

- **Exact RTA**. Since the improvement of this thesis focuses on the approximate RTA, the relation to exact RTA work is limited. One possibility would be that the fast analysis technique of storing points could also be applied to the exact variant. This would mean that points would have to be stored for the smallest periodic pattern which would be the entire LCM. One would have to further investigate the gain of such an approach.

- **Best case RTA**. Similarly, to exact RTA, best case RTA of Redell [RS02] is an exact approach. However, the work of this thesis and the work performed on best case RTA are orthogonal in the sense that they improve RTA in different and independent ways. The orthogonality also means that the improvements does not invalidate each other. And thus, RTA for distributed systems with precedence (offsets and release jitter) [TC94, PG98] can be further improved upon by applying the work of this thesis together with the work of best case response times [PGG98, RS02].

- **Improving RTA using precedence information**. The work of applying high level precedence information aims at eliminating cases that can not occur at run-time.

Our fast and tight improvements of higher priority task interference, statically stored imposed interference, however, zeroes in on a more detailed level of the analysis. Thus, the two methods are orthogonal. In fact, the two approaches are complementary, in the sense that the most gain of using precedence information is achieved when jitter is very high, whereas the imposed interference method gives the most gain when jitter is low. Hence, it is an attractive approach to combine them in a single approximate RTA method.

- **RTA for earliest deadline first systems**. The RTA method for EDF is very similar to the one of FPS scheduled systems. The approximate method is defined the same way as for the FPS case. It seems that both the approaches of fast and tight methods for FPS RTA could be straightforwardly incorporated into the EDF variant. However, one must ensure that the higher priority task interference exhibits a periodic pattern in order to store that statically. If that can be done the fast method could have an even greater impact on the analysis time of offset based RTA under EDF since the number of calculations of interference is higher than in the FPS case.

CHAPTER 6
Conclusions and future work

6.1 Conclusions

What distinguishes a real-time system from other computer systems, such as desktop systems, is how one views and deals with the notion of time. For real-time systems, correct temporal behavior is a vital part of the system's overall correctness criteria. When developing hard real-time and safety critical systems, the worst case temporal behavior must be guaranteed at design time.

The trend in developing embedded real-time systems is an ever increasing complexity in both the number of functions and the diversity of functionality [HMTN05]. In order to meet the challenges of reliability and safety requirements developers need to have supporting development methods and analysis tools at their disposal.

Response-Time Analysis (RTA) is able to predict and guarantee a system's worst case temporal behavior at design time. RTA is a widely applicable method, since it provides a schedulability test which is performed as a repeated step in several optimization problems. Examples include task allocation and attribute assignment, where a schedulability test is performed for every possible configuration.

In order for RTA to produce precise response times, the task model should reflect requirements on task constraints together with the run-time system's possibilities and limitations, as accurately as possible. The task model with offset aims at expressing temporal dependencies in task activations. Consequently, the pessimism of the basic RTA is reduced since the traditional critical instant of simultaneous release of all tasks is no longer possible. Tasks with offsets is a general task model that has been extended in many ways, including shared resources, jitter, arbitrary large or small task attributes, and non-preemptive tasks. Thus, tasks with offsets and it's corresponding RTA is able to accurately model many real situations, e.g., distributed and hybrid static and dynamic systems.

This thesis provides contributions in the area of RTA by extending the applicability of RTA for tasks with offsets:

- By further reducing the pessimism of calculated response times. We enable this by revealing and exploiting a misconception concerning higher priority task interference in the RTA formulae. A new concept, imposed interference, is introduced which is able to produce typically 15% shorter response times than previous approximate methods. A tight RTA has the benefit of not overestimating resources, and thus, is attractive for use in predicting the temporal behavior of embedded real-time systems.

- By introducing techniques that enable efficient implementations of the RTA method. The essence of these techniques concerns higher priority task interference. We recognize that this interference exhibits a repetitive and discretely increasing pattern that can be stored statically, and during fix-point calculations simply perform an efficient table lookup. Another technique which speeds up the analysis is to introduce pessimism when modeling higher priority task interference. If the pessimism is introduced in places where a fix-point cannot be reached, the fix-point calculation will converge in fewer iterations. The complexity of RTA is reduced, and thus, arbitrary large improvements over previous methods can be obtained by scaling up the task set size.

  However, with realistically sized task sets, simulations show that with the techniques presented in this thesis, RTA will be at least two orders of magnitude faster than with previous implementations. This indicates that RTA for tasks with offsets is efficient in being able to handle larger task sets such as those in real industrial applications.

- By combining the two independent improvements of fast and tight RTA, the traditional trade-off situation between a fast RTA on one hand, and accurate response times on the other, is practically eliminated. The resulting *tight and fast* method exhibits the most accurate response times as well as the fastest analysis time among all comparable approximate RTA methods.

- By illustrating how RTA for tasks with offset can be used in conjunction with commercially available tools for analyzing hybrid scheduled safety critical and hard real-time industrial systems. Being able to choose between static and dynamic scheduling on a per function basis instead of an overall system level, simplifies this design trade-off, and consequently the entire engineering process. Furthermore, the offset concept, viewed as a modeling and design concept, facilitates simplification of the development process, by modeling existing temporal dependencies, such as precedence relations. Offsets as a design concept, together with runtime support, extends the expressiveness and applicability of the task model by being able to introduce temporal dependencies among tasks. Thus, the offset concept extends the applicability of RTA for systems with complex and mixed constraints. A system having tight responsiveness, as well as control performance constraints, is an example of a system with such complex and mixed constraints.

The RTA method allows the timing behavior of embedded real-time systems to be accurately assessed at the design time of the system. The contributions of this thesis consist of techniques that extend the applicability of RTA by making them more eligible for integration in development tools for embedded real-time systems. Furthermore, we show how these contributions, taken together with previous and related work, can be instrumental in the development of predictable real-time systems.

6.2 Future work

Future work in the field of RTA can take many different directions. This section outlines, what I believe are the two most important ones. One relates specifically to the work of this thesis, while the other one applies to RTA in general:

- **Bring RTA for tasks with offsets into a real development context**. The work of this thesis has been verified, by simulation studies, on a conceptual level only. In order to validate the claims of the industrial relevance of this thesis, the presented improvement techniques must be implemented in commercially available development tools and used in full scale development projects. An ongoing research project called MultEx[1], in collaboration with an operating system and development tool vendor, Arcticus Systems [Arc], will address this issue. Arcticus Systems focuses their efforts on dependable safety critical applications.

[1]MultEx URL: http://www.mrtc.mdh.se/projects/multex/

MultEx aims at providing development mechanisms and multiple execution models that relieve some of the specification burden of the developer. In achieving this, RTA plays a major part. With implementation of RTA in Arcticus' development tools, real industrial settings can be set up. And hopefully, in the long run, apply RTA in a real, full scale, development project.

- **Further reducing the pessimism of RTA**. The complexity of embedded real-time applications is growing in both size and diversity. For RTA to be useful for such applications, task models must be able to express the requirements of functionality as well as the possibilities and constraints of the run-time systems, as accurately as possible. Tasks with offsets, and the corresponding RTA, reduce the pessimism by being able to express and introduce temporal dependencies among tasks. The pessimism is further reduced by techniques presented in this thesis. Other techniques, reducing the pessimism, include best case response time analysis [RS02, PGG98] and utilizing precedence information [Red04, PG99] among tasks. However, Wall *et al.* recognize that traditional real-time analysis models, such as RTA, are not applicable for large and complex real-time systems. The existing models are too simple to accurately capture the systems temporal behavior, resulting in a too pessimistic analysis [WAN$^+$03].

  We believe that in order for RTA to be useful for large systems that have many mutual task dependencies, further studies of sources of pessimism must be conducted. An example of such a study would be to identify and express dependencies in WCETs of tasks belonging to the same transaction, instead of assuming WCET for each an every one of them.

BIBLIOGRAPHY

[ABRW91] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, May 1991.

[ABT+93] N.C. Audsley, A. Burns, K. Tindell, M.F. Richardson, and A.J. Wellings. Applying New Scheduling Theory to Static Priority Preemptive Scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[Arc] Arcticus Systems Home-Page. http://www.arcticus.se.

[Aud91] N.C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times. Technical Report YCS-164, Dept. of Computer Science, University of York, England, November 1991. Available at ftp://ftp.cs.york.ac.uk/pub/-realtime/papers/YCS164.ps.Z.

[BB99] I. Bate and A. Burns. An Approach to Task Attribute Assignment for Uniprocessor Systems. In *Proc. of the 11th Euromicro Workshop of Real-Time Systems*, June 1999.

[BNC03] I. Bate, P. Nightingale, and A. Cervin. Establishing timing requirements and control attributes for control loops in real-time systems. In *Proc. of the 15th Euromicro Conference on Real-Time Systems*, July 2003.

[BTW95] A. Burns, K. Tindell, and A Wellings. Effective Analysis for Engineering Real-Time Fixed Priority Schedulers. *IEEE Transactions on Software Engineering*, 22(5):475–480, May 1995.

[But97] G.C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, 1997. ISBN 0-7923-9994-3.

[Cer99] A. Cervin. Improved scheduling of control tasks. In *Proc. of the 11th Euromicro Workshop of Real-Time Systems*, pages 4 – 10, June 1999.

[EEN^{+}03] Jakob Engblom, Andreas Ermedahl, Mikael Nolin, Jan Gustafsson, and Hans Hansson. Worst-case execution-time analysis for embedded real-time systems. *Journal of Software Tool and Transfer Technology (STTT)*, 4(4), 8 2003.

[EHS97] A. Ermedahl, H. Hansson, and M. Sjödin. Response-Time Guarantees in ATM Networks. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS)*, pages 274–284. IEEE Computer Society Press, December 1997. URL: http://www.docs.uu.se/~mic/-papers.html.

[Erm03] A. Ermedahl. *A Modular Tool Architecture for Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Uppsala University, Department of Information Technology, 2003.

[Flx] FlexRay Home Page. http://www.flexray-group.org/.

[GG95] J.J. Gutiérrez García and M. González Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *3rd Workshop onParallel and Distributed Real-Time Systems*, April 1995.

[HMTN05] Kaj Hänninen, Jukka Mäki-Turja, and Mikael Nolin. Industrial Requirements in Development of Embedded Real-Time Systems – Interviews with Senior Designers. In *WiP Session of 17th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2005.

[JP86] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[KAS93] D.I. Katcher, H. Arakawa, and J.K. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Transactions on Software Engineering*, 19(9):920–934, September 1993.

[KRP+99] M.H. Klein, T. Ralya, Bill Pollak, R. Obenza, and M.G. Harbour. *A Practitioners Handbook for Real-Time Analysis*. Kluwer Academic Publishers, fifth edition, 1999. ISBN 0-7923-9361-9.

[Leh90] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. 11th IEEE Real-Time Systems Symposium (RTSS)*, pages 201–212, December 1990.

[LL73] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[Loc92] C.D. Locke. Software Architecture For Hard Real-Time Applications - Cyclic Executives vs. Fixed Priority Executives. *The Journal of Real-Time Systems*, 4:37–53, 1992.

[NGS+01] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, and N. E. Bånkestad. Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry. In *Eigth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*. IEEE Computer Society, April 2001.

[PG98] J.C. Palencia Gutiérrez and M. González Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[PG99] J.C. Palencia Gutiérrez and M. González Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 328–339, December 1999.

[PG03] J.C. Palencia Gutiérrez and M. González Harbour. Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF. In *Proc. of the 15th Euromicro Conference on Real-Time Systems*, June 2003.

[PGG98] J.C. Palencia Gutiérrez, J.J. Gutiérrez García, and M. González Harbour. Best-Case Analysis for Improving the Worst Case Schedulability Test for Distributed Hard Real-Time Systems". In *Proc. of the 10th Euromicro Workshop of Real-Time Systems*, June 1998.

[Pun97] S. Punnekkat. *Schedulability Analysis for Fault Tolerant Real-time Systems*. PhD thesis, University of York, June 1997.

[Red04] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *Proc. of the 16th Euromicro Conference on Real-Time Systems*, June 2004.

[RS02] O. Redell and M. Sanfridsson. Exact Best-Case Response Time Analysis of Fixed Priority Scheduled Tasks. In *Proc. of the 14th Euromicro Conference on Real-Time Systems*, June 2002.

[RSL88] R. Rajkumar, L. Sha, and J.P. Lehoczky. Real-time Synchronisation protocols for Multiprocessors. In *Proc. 9th IEEE Real-Time Systems Symposium (RTSS)*, pages 259–269, December 1988.

[RT02] O. Redell and M. Törngren. Calculating Exact Worst-Case Response Times for Static Priority Scheduled Tasks with Offsets and Jitter. In *Proc. 8th IEEE Real-Time Technology and Applications Symposium (RTAS)*, September 2002.

[SAr^{+}04] L. Sha, T. Abdelzaher, K-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, 28(2/3):101–155, 2004.

[SH98] M. Sjödin and H. Hansson. Improved Response-Time Calculations. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998. URL: http://www.docs.uu.se/~mic/papers.html.

[Sjö00] Mikael Sjödin. *Predictable High-Speed Communications for Distributed Real-Time Systems*. PhD thesis, Uppsala University, Dept. of Information Technology, May 2000.

[SN02] Kristian Sandström and Christer Norström. Managing Complex Temporal Requirements in Real-Time Control Systems. In *9th IEEE Conference on Engineering of Computer-Based Systems*, April 2002.

[SRL87] L. Sha, R. Rajkumar, and J.P. Lehoczky. Task scheduling in distributed real-time systems. In *IEEE Industrial Electronics Conference*, 1987.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: an Approach to Real Time Synchronization . *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[TB94] K. Tindell and A. Burns. Fixed Priority Scheduling of Hard Real-Time Multimedia Disk Traffic. *The Computer Journal*, 37(8):691–697, 1994.

[TC94] K. Tindell and J. Clark. Holistic Schedulability Analysis For Distributed Hard Real-Time Systems. Technical Report YCS197, Real-Time Systems Research Group, Department of Computer Science, University of York, November 1994. URL ftp://-ftp.cs.york.ac.uk/pub/realtime/papers/YCS197.ps.Z.

[THW94] K. Tindell, H. Hansson, and A. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proc. 15th IEEE Real-Time Systems Symposium (RTSS)*, pages 259–263. IEEE, IEEE Computer Society Press, December 1994.

[Tin92a] K. Tindell. An extendible approach for analyzing fixed priority hard real-time tasks. Technical Report YCS189, Dept. of Computer Science, University of York, England, 1992.

[Tin92b] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

[Tur02] J. Turley. The two percent solution. *Embedded systems Programming (Embedded.com)*, December 2002. http://www.embedded.-com/showArticle.jhtml?articleID=9900861.

[Vol] Volvo Construction Equipment. http://www.volvoce.com.

[WAN$^+$03] A. Wall, J. Andersson, J. Neander, C. Norström, and M. Lembke. Introducing temporal analyzability late in the lifecycle of complex real-time systems. February 2003.

[XP00] J. Xu and D.L. Parnas. Priority Scheduling Versus Pre-Run-Time Scheduling. *The Journal of Real-Time Systems*, 18(1):7–23, January 2000.

II

Included Papers

CHAPTER 1

Paper A: Tighter Response-Times for Tasks with Offsets

JUKKA MÄKI-TURJA AND MIKAEL NOLIN

In the proceedings of Real-Time and embedded Computing Systems and Applications conference (RTCSA).

GOTHENBURG SWEDEN, AUGUST 2004.

Abstract

We present an improvement to the analysis methods for calculating approximate response times for tasks with offsets. Our improvement calculates tighter (i.e. lower) response times than does earlier approximation methods, and simulations show that the method, under certain conditions, calculates the exact worst-case response time.

We reveal, and exploit, a misconception in previous methods concerning the interference a higher priority task poses on a lower priority task. In this paper we show how the generally accepted concept of “released for execution” interference produces unnecessary pessimistic response times for the approximate response-time analysis (RTA), presented by Tindell [Tin92] and Palencia Gutiérrez *et al.* [PG98]. This concept of interference does not cause any pessimism in response-time analysis for tasks without offsets (neither in the exact analysis with offsets), and has thus remained undetected over the years.

Instead, we propose the concept of “imposed” interference, which more accurately captures the interference a task causes a lower priority task. We provide formal proofs that “imposed” interference is never higher than “released for execution” interference and that it never underestimates the interference caused by higher priority tasks. We also show, by simulations on randomly generated task sets, that our improvement results in response times that outperform previous approximate methods. A typical improvement results in about 12% better admission probability (more than 30% under certain circumstances can be obtained).

1.1 Introduction

A powerful and well established schedulability analysis technique is the *Response-Time Analysis* (RTA) [ABD$^+$95]. RTA is applicable to systems where tasks are scheduled in priority order which is the predominant scheduling technique used in real-time operating systems today. RTA is a method to calculate worst-case response times for tasks in hard real-time systems. Hence, RTA can be used to perform schedulability tests, i.e., testing if tasks in a system will meet their deadlines.

In this paper, we reveal and exploit a misconception concerning the interference a task causes a lower priority *task under analysis*, one of the core concepts of RTA. The essence of this misconception is that the amount of interference a higher priority task is causing is occasionally overestimated. The misconception has its origin in the original RTA presented by Joseph and Pandya [JP86] for Liu and Layland's classical task model [LL73]. In essence, Joseph and Pandya's RTA simulates the amount of execution-time queued in the ready-queue of an operating system, i.e. when a (higher priority) task is released for execution, its execution time is added to the response time of the task-under analysis. Hence, we call this concept for "released for execution" interference. For traditional RTA, for tasks without offsets, this concept will not cause any overestimation of calculated response times. However, as we will show in this paper, this concept is an overestimation of the interference, and when performing approximate RTA for task with offsets, it results in unnecessary pessimistic response times.

Accounting for offsets between tasks gives significantly tighter response times than using the traditional notion of a critical instant where all tasks in a system are considered to be released simultaneously [LL73]. In fact, many systems that will be deemed infeasible by RTA without offsets will be feasible when taking offsets into account. The first RTA for tasks with offsets was presented by Tindell [Tin92]. He provided an exact algorithm for calculating response time for tasks with offsets. However, this algorithm becomes computationally intractable for anything but small task sets due to its exponential time complexity. In order to deal with this problem, Tindell provided an approximation algorithm, with polynomial complexity, which gives pessimistic, but safe results (worst case response times are never underestimated).

Several researchers have extended the work provided by Tindell. In this paper we focus on the approximate analysis, which was generalized and formalized by Palencia Gutiérrez *et al.* [PG98]. They introduced dynamic offsets, allowed offsets and deadlines larger than period, and made some improvement of the approximation algorithm. Palencia Gutiérrez *et al.* also provided improvements in order to calculate tighter response times in certain situations [PG99]. Redell further improved their work by giving a method to calculate even lower response times [Red03].

However, both improvements [PG99, Red03] are only useful in very special circumstances where task priorities are chosen in a particular way and task jitter is extremely high[1]. Hence, their improvements are of limited generality. The focus of their methods is on finding infeasible execution orders between tasks and removing these execution orders from the set of possible critical instants. The method we present in this paper is more general and can straight forwardly be combined with the above described improvements. In fact, our approach presented here is complementary to these approaches in the sense that the most improvement for our method is achieved when jitter is low.

In this paper we present a novel interpretation of higher priority task interference: "imposed" interference, with corresponding changes to the response-time formulae, which will result in less pessimistic response times for tasks with offsets using the approximation algorithm. We formally prove that response times obtained with this novel method are never greater than the method presented by Palencia Gutiérrez *et al.* [PG98]. Furthermore, we also show that our method does this without the risk of ever underestimating response times.

To quantify the improvements gained with our method we present an evaluation, showing that with our method presented in this paper, one can typically gain about 15% lower response times in over 50% of the cases, resulting in a 12% higher admission probability, compared to existing approximate methods. In more extreme cases (just one transaction) about 30% higher admission probability can be obtained.

Paper Outline: In section 1.2 we present the pessimistic "released for execution" interference and introduce our novel concept of "imposed" interference. In section 1.3 we revisit and restate the original offset RTA [Tin92, PG98]. In section 1.4 we modify this RTA to use the concept of "imposed" interference instead, and show some consequences and proofs of correctness. Section 1.5 presents evaluations of our method, and finally, section 1.6 concludes the paper and outlines future work.

[1]Priority needs to be chosen so that transactions can "interlock" each other, and the jitter needs to be in parity with, or greater than, the task's periods. Otherwise the proposed improvements will have little or no effect.

1.2 The Concept of Interference

Classical response-time analysis for Liu and Layland's periodic task model [LL73] (where a task τ_i has a period T_i and worst-case execution-time C_i), presented first by Joseph and Pandya [JP86], states that the worst case response time, for a task under analysis (τ_i), occurs when it is released at the same time as all higher priority tasks. Under this assumption the worst case response time, R_i is:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \mathit{interference}_j(R_i) \tag{1.1}$$

where C_i is the execution-time of task i, $hp(i)$ is the set of higher priority tasks, and *interference*$_j(t)$ is the amount of interference task j causes during time-interval t. The interference formula presented by Joseph and Pandya is [JP86]:

$$\mathit{interference}_j(t) = \left\lceil \frac{t}{T_j} \right\rceil C_j$$

where the ceiling expressions calculates the number of instances of task j. Here the full interference on each task instance (C_j) occurs immediately when the task is released. We denote this concept of interference as "released for execution" interference.

This, however, is an overestimation of the interference that τ_i actually can experience. In fact, the interference *experienced* by τ_i during a time interval can never exceed the size of the time interval. Or more precisely, the interference experienced can never grow faster than the considered interval. Formally, the derivative of the interference cannot be greater than the derivative of the time interval:

$$\frac{\mathrm{d}\mathit{interference}_j(t)}{\mathrm{d}t} \leq \frac{\mathrm{d}t}{\mathrm{d}t} \qquad \Rightarrow \qquad \frac{\mathrm{d}\mathit{interference}_j(t)}{\mathrm{d}t} \leq 1 \tag{1.2}$$

THEOREM 1.1 *Consider a task τ_j, activated at time 0 and subsequently with period T_j, having execution-time C_j ($0 < C_j \leq T_j$). For a positive time-interval $t = kT_j + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_j$), $kC_j + \min(t', C_j)$ is an upper bound on the interference τ_j can impose on any lower priority task during t.*

PROOF OF THEOREM 1.1 *During kT_j, τ_j imposes an amount of interference of kC_j (task instances are activated periodically), one instance for every* *period. During the remaining time interval, t', τ_j can, according to equation 1.2, never impose more interference than the length of the interval itself. Hence, $kC_j + t'$ is an upper bound on the interference τ_j, can impose during t.*

However, the last instance of τ_j (when activated $t' = 0$), cannot contribute with more interference than its execution time C_j. Hence, $kC_j + C_j$ is also an upper bound on the interference τ_j can impose during t.

Combining these upper bounds (by taking the minimum of them) we get $kC_j + \min(t', C_j)$ as an upper bound on the interference τ_j can impose during t. □

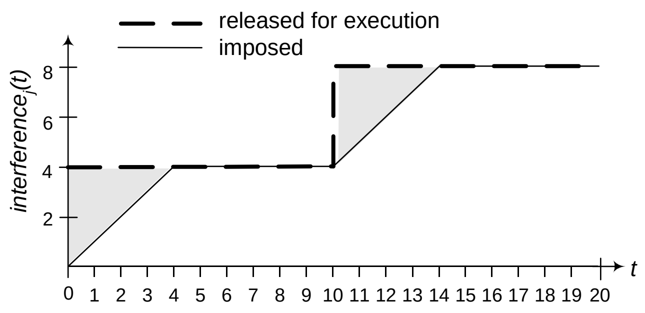


Figure 1.1: Released for execution vs. imposed interference

We denote the concept of interference which is bounded by *interference*$_j(t)$ and theorem 1.1 with "imposed" interference. As an example, consider a task with $T_j = 10$ and $C_j = 4$. Figure 1.1 illustrates the difference between "released for execution" and "imposed" interference for $t \in 0 \ldots 20$. The released for execution interference increases in a *stepped stair* fashion, whereas the imposed interference increases in a *slanted stair* fashion (with a derivative of 1 in the slants).

In figure 1.1 the shaded areas represent the overestimation made by the released for execution concept. For classical response-time analysis this overestimation has no effect on the calculated response time, and Joseph and Pandya's equation does yield exact worst case response times. The reason for this is that the response-time analysis calculation (which is done by fix-point iteration) has no solutions in the shaded areas (as discussed further in section 1.4.3). Also for exact RTA of task with offsets [Tin92] this overestimation does not yield any pessimism in the calculated response times.

1.3 Existing offset RTA

This section revisits the existing response-time analysis for tasks with offsets [Tin92, PG98] and illustrates some intuition behind the analysis and the formulae.

1.3.1 System model

The system model used is as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i (for non-periodic events T_i denotes the minimum inter-arrival time between two consecutive events). The activating events are mutually independent, i.e., phasing between them is arbitrary. A transaction, Γ_i, contains $|\Gamma_i|$ tasks, and each task is activated (released for execution) when a relative time, *offset*, elapses after the arrival of the external event.

We use τ_{ij} to denote a task. The first subscript denotes which transaction the task belongs to, and the second subscript denotes the number of the task within the transaction. A task, τ_{ij}, is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The system model is formally expressed as follows:

$$\begin{aligned}
\Gamma :=& \{\Gamma_1, \ldots, \Gamma_k\} \\
\Gamma_i :=& \langle \{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i \rangle \\
\tau_{ij} :=& \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} \rangle
\end{aligned}$$

There are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

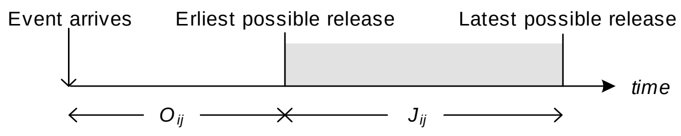


Figure 1.2: Relation between an event arrival, offset, jitter and task release

The relation between event arrival, offset, jitter and task release is graphically visualized in figure 1.2. After the arrival of the event the task τ_{ij} is never released for execution until its offset (O_{ij}) has elapsed. The release may be delayed by jitter (maximally until $O_{ij} + J_{ij}$) making its exact release uncertain.

For a more extensive explanation of task parameters see [PG98]. Parameters for an example transaction (Γ_i) with two tasks (τ_{i1}, τ_{i2}) are depicted in figure 1.3.

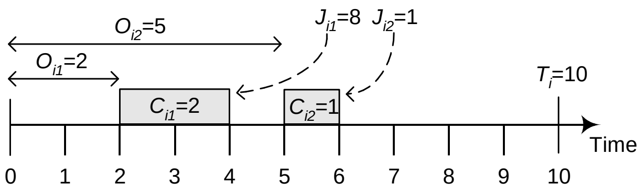


Figure 1.3: An example transaction Γ_i

1.3.2 Response-time analysis

The goal of RTA is to facilitate a schedulability test for each task in the system by calculating an upper bound on its worst case response time. We use τ_{ua} (task a, belonging to transaction Γ_u) to denote the *task under analysis*, i.e., the task who's response time we are currently calculating.

In the classical RTA (without offsets) the *critical instant* for τ_{ua} is when it is released at the same time as all higher (or equal) priority tasks [JP86, LL73]. In a task model with offsets this assumption yields pessimistic response times since some tasks can not be released simultaneously due to offset relations. Therefore, Tindell [Tin92] relaxed the notion of critical instant to be:

> At least one task in every transaction is to be released at the critical instant. (Only tasks with priority higher or equal to τ_{ua} are considered.)

Since it is not known which task that coincides with (is released at) the critical instant, every task in a transaction must be treated as a *candidate* to coincide with the critical instant.

Tindell's exact RTA tries every possible combination of candidates among all transactions in the system. This, however, becomes computationally intractable for anything but small task sets (the number of possible combinations of candidates is m^n for a system with n transactions and with m tasks per transaction). Therefore Tindell provided an approximate RTA that still gives good results but uses one single approximation function for each transaction. Palencia Gutiérrez *et al.* [PG98] formalized and generalized Tindell's work.

We will in this paper use the more general formalism of Palencia Gutiérrez *et al.*, although our proposed method is equally applicable to Tindell's original algorithm.

1.3.3 Interference function

Central to RTA is to capture the interference a higher or equal priority task (τ_{ij}) causes the task under analysis (τ_{ua}) during an interval of time t. Since a task can interfere with τ_{ua} multiple times during t, we have to consider interference from possibly several *instances*. The interfering instances of τ_{ij} can be classified into two sets:

Set1 Activations that occur before or at the critical instant and that can be delayed by jitter so that they coincide with the critical instant.

Set2 Activations that occur after the critical instant

When studying the interference from an entire transaction Γ_i, we will consider each task, $\tau_{ic} \in \Gamma_i$, as a *candidate* for coinciding with the critical instant.

RTA of tasks with offsets is based on two fundamental theorems:

1. The worst case interference a task τ_{ij} causes τ_{ua} is when *Set1* activations are delayed by an amount of jitter such that they all occur at the critical instant and the activations in $Set2$ have zero jitter.

2. The task of Γ_i that coincide with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

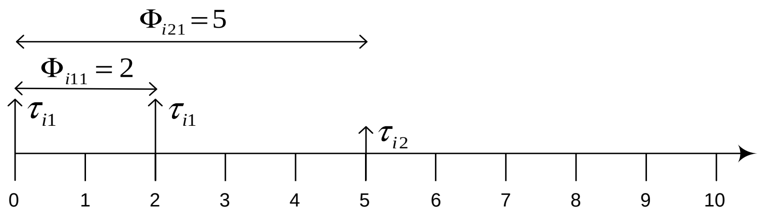
The phasing between a task, τ_{ij}, and a critical instant candidate, τ_{ic}, becomes (slightly reformulated compared to [PG98], see Appendix 1.6):

$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \mod T_i \tag{1.3}$$

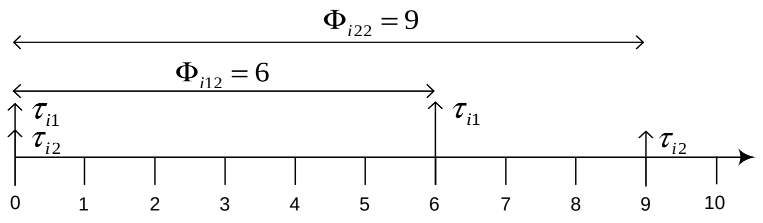
From the second theorem we get that τ_{ic} will coincide with the critical instant after having experienced its worst case jitter delay, i.e., the critical instant will occur at $(O_{ic} + J_{ic}) \mod T_i$, relative to the start of Γ_i. From this, the definition of Φ_{ijc} follows in order to keep the relative phasing (of releases) among tasks within Γ_i. An implication of this is that the first instance of a task τ_{ij} in *Set2* will be released at Φ_{ijc} time units after the critical instant, and subsequent releases will occur periodically every T_i.

Figure 1.4 illustrates the four different Φ_{ijc}-s that are possible for our example transaction in figure 1.3. The upward arrows denote task releases. The height of the upward arrows denotes the amount of execution released.

Figure 1.4(a) shows for the case that τ_{i1} coincides with the critical instant, the invocations in *Set1* (arriving at time 0) and the first invocations in *Set2*. Figure 1.4(b) shows the corresponding situation when τ_{i2} is the candidate to coincide with the critical instant.



(a) $\tau_{ic} = \tau_{i1}$



(b) $\tau_{ic} = \tau_{i2}$

Figure 1.4: Φ-s for the two candidates in Γ_i

Given the two sets of task instances (*Set1* and *Set2*) and the corresponding phase relative to the critical instant (Φ_{ijc}), the interference caused by task τ_{ij} can be divided into two parts:

1. The part caused by instances in *Set1* (which is independent of the time interval t), I_{ijc}^{Set1}, and
2. the part caused by instances in *Set2* (which is a function of the time interval t), $I_{ijc}^{\mathit{Set2}}(t)$.

These are defined as follows:

$$I_{ijc}^{\mathit{Set1}} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij} \qquad I_{ijc}^{\mathit{Set2}}(t) = \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij} \tag{1.4}$$

The interference transaction Γ_i poses on τ_{ua}, during a time interval t, when candidate τ_{ic} coincides with the critical instant, is:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t) \right) \tag{1.5}$$

Where $hp_i(\tau_{ua})$ denotes tasks belonging to transaction Γ_i, with priority higher or equal to the priority of τ_{ua}.

1.3.4 Approximation function

Since we beforehand cannot know which task, in each transaction, coincides with the critical instant, the exact analysis tries every possible combination [Tin92, PG98]. However, since this is computationally intractable for anything but small task sets, the approximate analysis defines one single, upward approximated, function for the interference caused by transaction Γ_i [Tin92, PG98]:

$$W_i^*(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t) \tag{1.6}$$

That is, $W_i^*(\tau_{ua}, t)$ simply takes the maximum of each interference function (for each candidate τ_{ic}).

As an example, consider again transaction Γ_i depicted in figure 1.3. Figure 1.5 shows the interference function for the two candidates (W_{i1} and W_{i2}), and it shows how W_i^* is derived from them by taking the maximum of the two functions at every t.

Given the interference (W_i^*) each transaction causes, during a time interval of length t, the response time of τ_{ua} (R_{ua}) can be calculated. Appendix 1.6 shows how to perform these response-time calculations.

1.4 Tight offset RTA

We begin this section with an illustrative example of how the original analysis overestimates the response time. Consider a simple transaction Γ_i depicted in figure 1.6 where jitter (J_{ij}) and blocking (B_{ij}) is zero.

Also consider a lower priority task, τ_{ua}, which is the single task in transaction Γ_u, with $C_{ua} = 2$. For this simplified task model where $B_{ij} = J_{ij} = 0$, $D_{ua} \leq T_u$ only one instance of the task under analysis is active at any point in time. This means that the response-time formulae, for the single lower priority

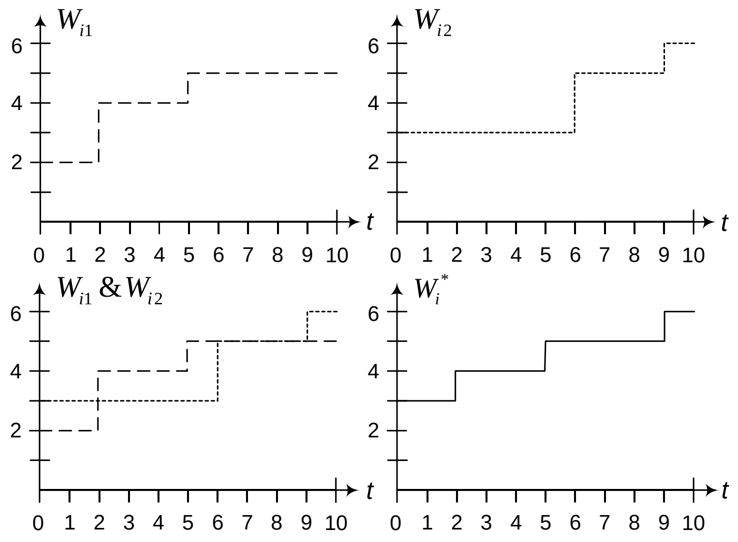


Figure 1.5: $W_{ic}(\tau_{ua}, t)$ and $W_i^*(\tau_{ua}, t)$ functions

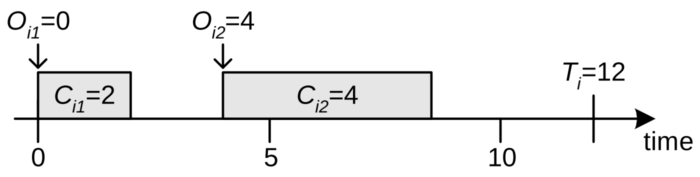


Figure 1.6: A simple example transaction

task, presented in appendix 1.6, can be reduced and simplified to:

$$R_{ua} = C_{ua} + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, R_{ua}) \tag{1.7}$$

The response-time calculation is performed by means of fix-point iteration (starting with $R_{ua} = 0$) as follows:

Iter#	t	W_{i1}	W_{i2}	W_i^*	R_{ua}
0					0
1	0	0	0	0	2
2	2	2	4	4	6
3	6	6	4	6	8
4	8	6	4	6	8

Where column "Iter#" denotes the iteration number in the fix-point iterations, "t" the time interval, "W_{i1}" and "W_{i1}" denotes $W_{ic}(\tau_{ua}, t)$ for the two candidate tasks τ_{i1} and τ_{i2} respectively. "W_i^*" denotes the value of $W_i^*(\tau_{ua}, t)$, and "R_{ua}" the calculated response time for the iteration. In iteration number 4 the fix-point iteration terminates (R_{ua} has the same value as in the previous iteration), and the calculated response time is $R_{ua} = 8$. However, it can easily be seen that a task with $C_{ua} = 2$ can never be preempted by both tasks τ_{i1} and τ_{i2} since both tasks are separated by at least 2 units of idle time. Hence, the actual worst case response time is $R_{ua} = 6$ and the response time is overestimated.

1.4.1 Using Imposed Interference

One property of the ceiling expression of $I_{ijc}^{\mathit{Set2}}(t)$ in equation 1.4 is that it returns the amount of interference "released for execution" at time t. This result in a *stepped stair* interference function. If we modify $I_{ijc}^{\mathit{Set2}}(t)$ in equation 1.4 so that it returns the interference "imposed" on τ_{ua} we get a *slanted stair* function (as proposed in section 1.2). The two slanted stair functions for our simple example transaction from figure 1.6 are shown in figures 1.8(a) and 1.8(b).

The slanted stairs are obtained by modifying $I_{ijc}^{\mathit{Set2}}(t)$ defined in equation 1.4 so that the "last" task instance, of the periodically activated tasks in *Set2*, does not interfere with its full execution time unless the interval t is sufficiently large. Our redefined version of $I_{ijc}^{\mathit{Set2}}(t)$ is:

$$
\begin{aligned}
I_{ijc}^{\textit{Set2}}(t) &= \left\lceil \frac{t^*}{T_i} \right\rceil C_{ij} - x \\
x &= \begin{cases} C_{ij} - (t^* \mod T_i) & \text{if } t^* > 0 \wedge \left(0 < t^* \mod T_i < C_{ij}\right) \\ 0 & \text{otherwise} \end{cases} \\
t^* &= t - \Phi_{ijc}
\end{aligned}
\tag{1.8}
$$

where Φ_{ijc} is defined in equation 1.3 and x is used to generate the slants of the "imposed" interference function. Figure 1.7(a) illustrates a sequence of task releases, and figure 1.7(b) shows how the value of x varies accordingly.

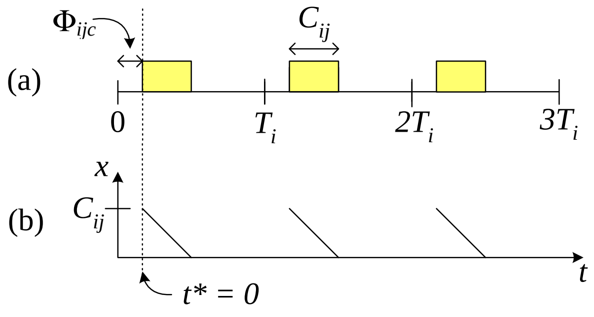


Figure 1.7: Relation between task release and x

The slanted stair functions, generated by equation 1.8, are depicted in figures 1.8(a) and 1.8(b). Figure 1.8(c) shows them overlaid. Using our new version of $I_{ijc}^{\textit{Set2}}(t)$ in equation 1.5 we get the maximized slanted stairs interference function, representing the approximation function W_i^*, shown in figure 1.8(d).

With the new definition of interference in equation 1.8 we can now use equation 1.7 to calculate a new response time R_{ua} for our example as follows:

Iter#	t	W_{i1}	W_{i2}	W_i^*	R_{ua}
0					0
1	0	0	0	0	2
2	2	2	2	2	4
3	4	2	4	4	6
4	6	4	4	4	6

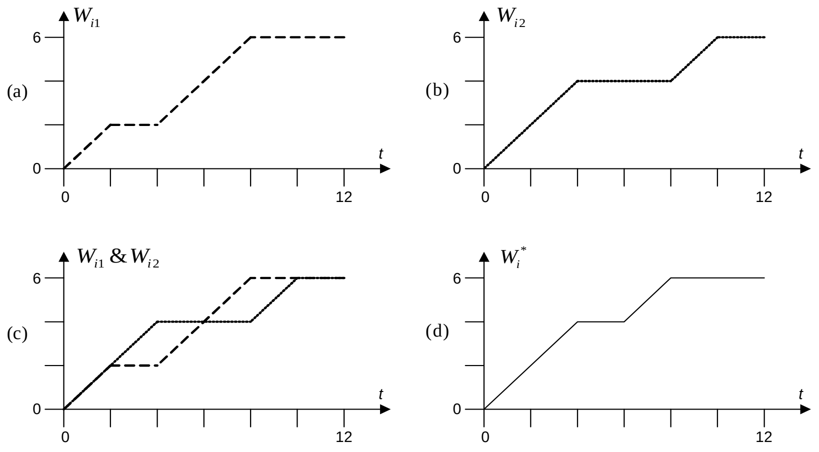


Figure 1.8: Interference *imposed* by our example transaction

We note that our new definition of $I_{ijc}^{Set2}(t)$ makes the analysis able to "see" the empty slot between tasks τ_{i1} and τ_{i2}, something the original analysis overlooked.

Hence, the calculated response time (6) is lower than that of the original analysis (8), and in section 1.5 we will quantify this improvement in more general terms.

1.4.2 Correctness Criteria

For our proposed modification to $I_{ijc}^{Set2}(t)$ in equation 1.8 to be correct, and not produce greater response times than the original analysis, three criteria have to be fulfilled:

- The new definition of $I_{ijc}^{Set2}(t)$ is not allowed to be greater than the old definition (for any t). If this condition holds, the analysis performed with the new definition is guaranteed not to yield larger response times than the old definition does.

- The new definition of $I_{ijc}^{Set2}(t)$ must not underestimate the interference caused by *Set2*-tasks. If the interference is underestimated, analysis performed with the new definition could yield unsafe response-time estimates.

- The new definition of $I^{\mathit{Set2}}_{ijc}(t)$ must yield a monotonically increasing interference function $W_i^*(\tau_{ua}, t)$. Monotonicity is required to guarantee that at least one solution to the response-time formula exists and that fix-point iteration finds the smallest existing solution [SH98].

THEOREM 1.2 *For a given task under analysis, τ_{ua}, and one candidate task, $\tau_{ic} \in \Gamma_i$, our new definition of I^{Set2}_{ijc} (equation 1.8) is never greater than the old definition (equation 1.4).*

PROOF OF THEOREM 1.2 *x, as defined in equation 1.8, is used to decrease the calculated value of I^{Set2}_{ijc}. Since x, by definition, is never negative, it can never contribute to making equation 1.8 greater than equation 1.4.* $\square$

THEOREM 1.3 *For any time interval $t \geq 0$ our new definition of $I^{\mathit{Set2}}_{ijc}(t)$ (equation 1.8) never underestimates the interference caused by Set2 task instances.*

PROOF OF THEOREM 1.3 *Set2 task instances arrive periodically (per definition) with period T_i, with the first instance arriving at Φ_{ijc}.*

We first treat the time before the first invocation in Set2, i.e. $t < \Phi_{ijc}$. During this time interval $t^* < 0$ and hence $x = 0$. Since, $t < \Phi_{ijc} < T_i$ then $t^* > -T_i$ and the ceiling expression in equation 1.8 evaluates to zero. Hence, the whole equation 1.8 is also zero. Since the interference before the first invocation obviously is zero, equation 1.8 does not underestimate the interference before the first invocation.

For times at or after the first invocation, i.e. $t \geq \Phi_{ijc}$, we have $t^* \geq 0$. Now, assume $t^* = kT_i + t'$, where $k \in \mathbb{N}$ and $0 \leq t' < T_i$ (the relation between t, t^* and t' is graphically visualised in figure 1.9). If the interference calculated by equation 1.8 is not below the safe upper bound *defined by theorem 1.1:*

$$\text{safe upper bound} = kC_{ij} + \min(t', C_{ij})$$

then the interference is not underestimated.

We divide the proof into three cases depending on the value of t' for a time-interval t (the three different cases are depicted graphically in figure 1.10):

- *$t' \geq C_{ij}$: The ceiling expression in equation 1.8 evaluates to $k + 1$ and the interference is thus $(k + 1)C_{ij} - x$. Further, when $t' \geq C_{ij}$ then $t^* \bmod T_i \geq C_{ij}$ resulting in $x = 0$, hence the interference is $(k + 1)C_{ij}$, which is not below* safe upper bound.

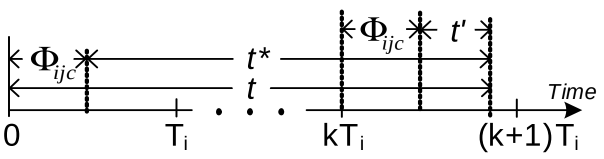


Figure 1.9: Relation between t, t^* and t'

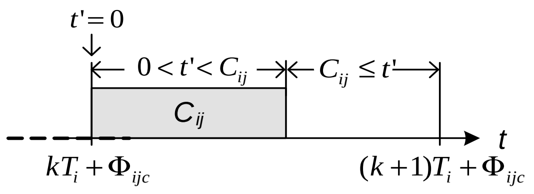


Figure 1.10: Three proof cases for t'

- $0 < t' < C_{ij}$*: The ceiling expression in equation 1.8 evaluates to* $k+1$ *and the interference is thus* $(k+1)C_{ij} - x = kC_{ij} + C_{ij} - x$*. Further, when* $0 < t' < C_{ij}$ *then* $0 < t^* \mod T_i < C_{ij}$ *and* $x = C_{ij} - t'$*, hence the interference is* $kC_{ij} + C_{ij} - (C_{ij} - t') = kC_{ij} + t'$*, which is not below* safe upper bound.

- $t' = 0$*: The ceiling expression in equation 1.8 evaluates to* k *and the interference is thus* $kC_{ij} - x$*. Further, when* $t' = 0$ *then* $t^* \mod T_i = 0$ *and* $x = 0$*, hence the interference is* kC_{ij}*, which is not below* safe upper bound *(since* $t' = 0$*).* □

THEOREM 1.4 *Our new definition of* $I_{ijc}^{Set2}(t)$ *(equation 1.8) is (non-strictly) monotonically increasing with the time interval* t*.*

PROOF OF THEOREM 1.4 *We prove this by showing that the derivative of equation 1.8 is never negative. First, we conclude that a negative derivative of* x *cannot contribute to make the derivative of equation 1.8 negative (since* x *is* subtracted *in equation 1.8). We also conclude that if* x *is disregarded (i.e. assumed to be 0), then equation 1.8 does not have a negative derivative in any point.*

We divide the proof into three cases, depending on the value of $t^* \mod T_i$ *for times* t*:*

- $t^* \mod T_i \geq C_{ij}$*: In this case x is continuously 0, hence the derivative of x is 0, and equation 1.8 cannot have a negative derivative.*

- $0 < t^* \mod T_i < C_{ij}$*: In this case the derivative of x is -1, hence the derivative of equation 1.8 cannot have a negative derivative.*

- $t^* \mod T_i = 0$*: For this case we conclude that equation 1.8 is continuous, since at time $t + \epsilon$ (for an arbitrary small and positive ϵ) the ceiling expression has increased with C_{ij} and x has increased with $C_{ij} - \epsilon$, hence equation 1.8 has increased with exactly ϵ. Thus, the derivative of equation 1.8 at such times t is 1.* $\square$

1.4.3 Discussion

At first glance, it is not directly obvious that lowering the interference function $W_{ic}(\tau_{ua}, t)$ should automatically give lower response times. In fact, the stepped-stair interference function has been used for many years to represent the interference in RTA [ABD$^+$95, ABT$^+$93], without introducing any pessimism.

The reason stepped stairs (in analysis without offsets) does not introduce pessimism can be found in our previous work [SH98]. In short, the fix-point iteration will terminate when the sum of all interference functions (demand) meets the line from origin with slope 1 (supply). Hence, replacing stepped stairs with slanted stairs (with slope 1) will not contribute to earlier fix-point convergence.

However, in approximate response-time analysis with offsets, the interference functions, W_{ic}-s, are not used directly in the fix-point iterations. Instead they are first subjected to a maximisation function (equation 1.6). This situation can be compared to floating point addition: if you round up the floating point numbers at each calculation step, instead of just in the end, you will loose precision. This corresponds to passing released for execution interference, instead of more precise imposed interference, to the maximisation function. Another view of this is that by using slanted-stair functions as input to the maximisation function, one essentially “delays” the time it takes for one low-interference scenario to overtake a high-interference scenario.

Figure 1.11(a) shows our simple example transaction from figure 1.6 with two arrows denoting the two possible scenarios for the critical instant (one “dashed” scenario and one “dotted” scenario). Figures 1.11(b) and 1.11(c) shows the stepped stairs and slanted stairs interference functions, respectively,

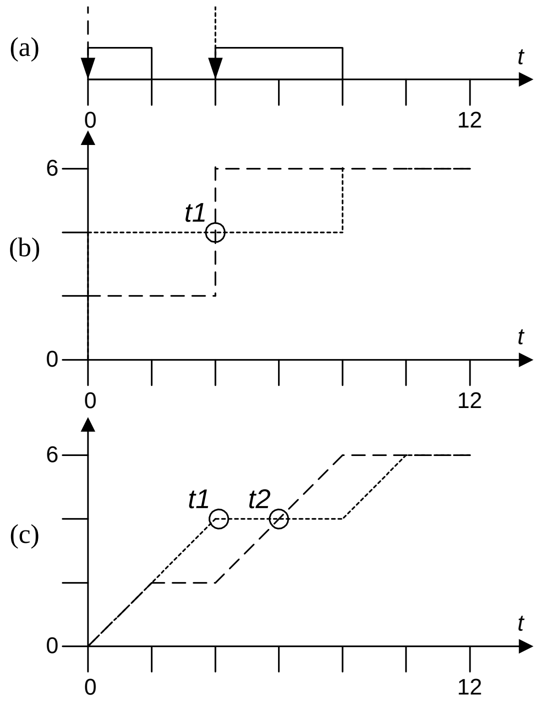


Figure 1.11: Stepped stairs vs. slanted stairs

for both scenarios. For times $t < t1$, the dotted scenario is the one with highest interference. Time $t1$ corresponds to the release of the second task in the dashed scenario. For the stepped stairs case, this means immediately adding another 4 units of interference to the dashed scenario, hence immediately making it the scenario with the highest interference. However, for the slanted stairs case, the time $t1$ means that the dashed line starts to increase, but not until time $t2$ it catches up with the dotted scenario. Hence, the interval between $t1$ and $t2$ represents the time by which the slanted stairs "delay" the dashed scenario to catch up with the dotted scenario. If fix-point convergence can be achieved during this interval, then RTA with imposed interference will calculate a lower response time than does RTA with released for execution interference.

1.5 Evaluation

In order to evaluate and quantify our proposed improvement, we have implemented the approximate response-time equations of appendix 1.6, using both the original definition of $I_{ijc}^{Set2}(t)$ from section 1.3 and our tighter version of $I_{ijc}^{Set2}(t)$ from section 1.4. Furthermore, we have also, as a comparison, implemented the exact analysis.

Using these implementations and a task-generator we have performed simulations of all three approaches by calculating the response time for a single low priority task, e.g., corresponding to an admission control situation.

1.5.1 Description of Task Generator

In our simulator we generate task sets that are used as input to the different implementations. The task-set generator takes the following parameters as input:

- Total system load (in % of total CPU utilization),
- The number of transactions to generate,
- The number of tasks per transaction to generate, and
- Jitter fraction (in % of the transaction periods).

Using these parameters a task set with the following properties is generated:

- The total system load is proportionally distributed over all transactions.

- Periods (T_i) are randomly distributed in the range 1.000 to 1.000.000 time units (uniform distribution).
- Each offset (O_{ij}) is randomly distributed within the transaction period (uniform distribution).
- The execution times (C_{ij}) are chosen as a fraction of the time between two consecutive offsets in the transaction. The fraction is the same throughout one transaction, and is selected so that the transaction load (as defined by the first property) is obtained.
- The jitter is set to the jitter fraction of the period ($J_{ij} = f * T_i$).
- Blocking (B_{ij}) is set to zero.
- The priorities are assigned in rate monotonic order [LL73].

1.5.2 Description of Simulation Setup

The heart of the improvement made to the approximate response-time analysis is a new definition of $I_{ijc}^{\mathit{Set2}}(t)$. We have implemented the response-time equations of appendix A which will show the effects of our improvements in a realistic scenario. However, neither interference from other tasks in Γ_u nor interference from previous instances of τ_{ua} comes into play in the admission control situation that we simulate. Taking interference from other tasks of Γ_u into account would yield less improvement of our methods, since W_i^* is not used for them (see appendix 1.6).

The setup of the simulation is as follows: a task set is generated according to input parameters (system load, number of tasks within a transaction, number of transactions, jitter). To simulate an admission control situation, we calculate the response time for a low priority task subjected to admission control.

We have calculated and compared the response times for our tighter analysis (Tight), Palencia Gutiérrez *et al.*'s original analysis (Orig) and the exact analysis (Exact). The results in section 1.5.3 have been obtained by taking the mean value from 1000 generated task-sets for each point in each graph. The graphs in the left and in the right columns also show the 95% confidence interval for these mean values.

We have measured three metrics from the simulations:

- "Admission probability (%)" — This metric measures the fraction of cases, out of the 1000 generated task sets, the admission control task passes the admission test (its response time is lower than its deadline).

- "Response-time improvement (%)" — This metric measures the average and maximum improvement (over Original) in response time for the task subjected to admission control. Improvement in response time for the tight analysis, R_{ua}^{Tight}, is defined as $1 - R_{ua}^{\text{Tight}}/R_{ua}^{\text{Orig}}$ (and analogous for the Exact analysis). Note that for this metric the original acts as baseline and thus only maximum and average improvement of (Tight) and (Exact) (over (Orig)) are plotted. Also note that the maximum value is one value (the maximum) out of 1000, which makes the behavior in these graphs statistically uncertain (they show what is possible without quantifying probability of occurrence).
- "Fraction of tasks with improvement (%)" — This metric measures the fraction of admission control tasks that results in a lower response time, compared to the original analysis (Orig). As for previous metric, the original approximate analysis is used as a baseline, hence no curve is plotted for that method. Note that this metric says nothing about the size of the improvements.

The first metric is to show what effect an improvement in response time could have in a realistic situation. The purpose of the last two metrics is to quantify the difference in response time between the three analysis methods.

1.5.3 Simulation Results

In the simulations we have varied our four task-generator parameters in different ways. Figures 1.12 to 1.15 show a subset of the simulation results. The exact analysis can only be run on small task sets; hence it is not present for larger tasks sets. For every parameter that is varied we show all three metrics described in the previous section, corresponding to top, middle, and bottom-most graph respectively, in each figure. (Note that, in the figures, "Tasks = x" denotes "x tasks/transaction".)

In all figures we start out at a base configuration where the number of tasks per transaction is 6, the number of transactions is 3, system load is 80% and the load of task under admission control is 2%. From this base configuration we vary the number of tasks/transaction (figure 1.12), number of transactions (figure 1.13), jitter (figure 1.14), while keeping the other parameters constant.

Figure 1.12 show the results when the number of tasks is varied between 1 and 13. For more than 5 tasks we can see, in the topmost graph, that the admission probability for (Tight) is around 12% higher than for (Orig). In middle graph we see that the average response-time improvement of (Tight)

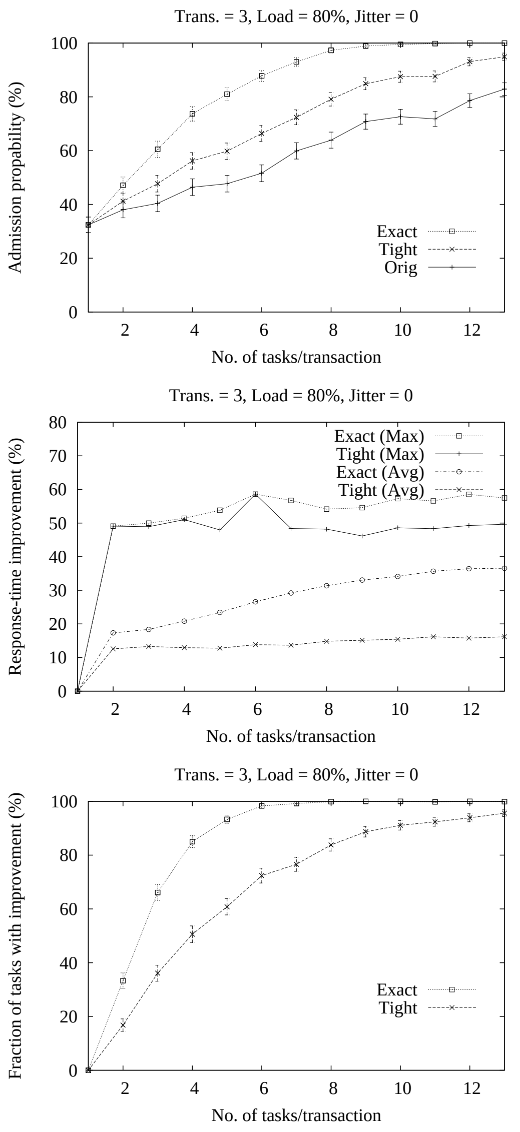


Figure 1.12: Varying the number of tasks per transaction

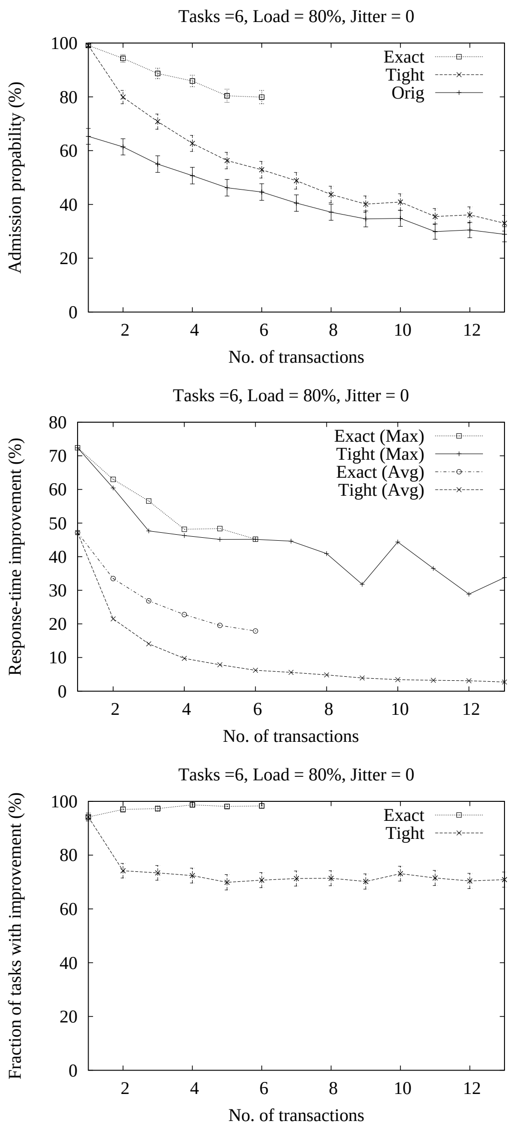


Figure 1.13: Varying the number of transactions

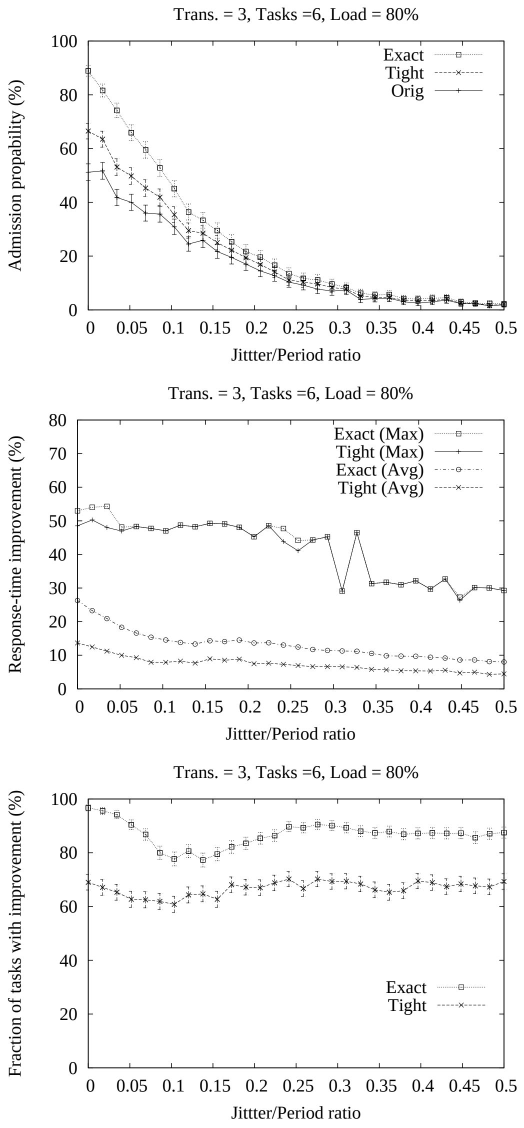


Figure 1.14: Varying the amount of jitter

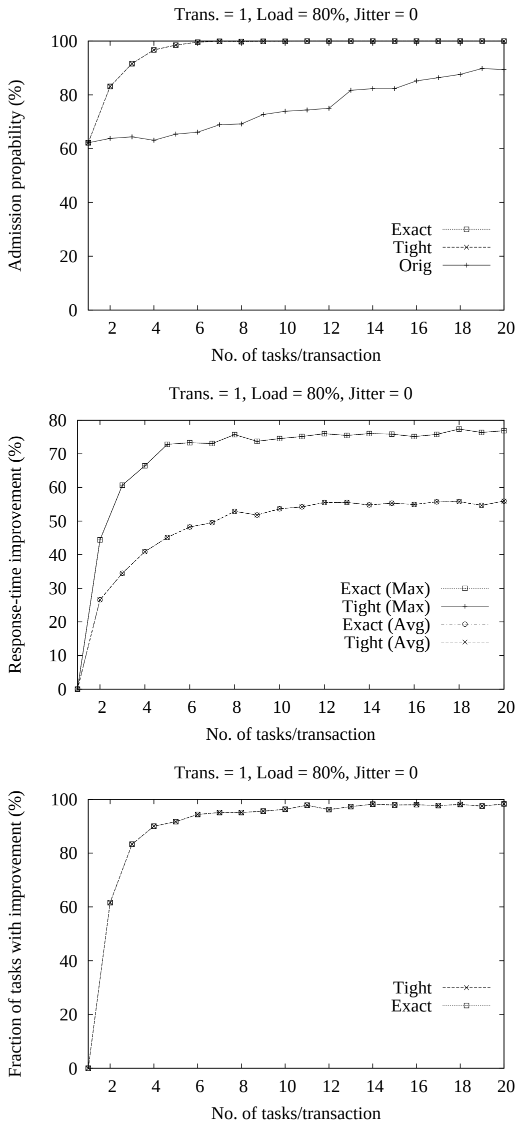


Figure 1.15: Hybrid scheduling scenario

is for 10 tasks over 15%, and that there are task sets (although rare) where improvement of more than 50% can be obtained. In the bottommost graph we see that when the number of tasks grows, so does the probability of a response-time improvement.

For figure 1.13, where the number of transactions is varied, a quite different picture emerges. The difference between (Orig) and (Tight) gets smaller as the number of transactions grows. This is not surprising, since in the case where the tasks/transaction ratio approaches 1, there are very few offset relations among tasks and the analysis approaches the analysis for tasks without offsets.

Figure 1.14 show what happens when jitter is varied. Not only does the admission probability decrease drastically, but also the relative improvement of (Tight) over (Orig). This is mainly due to the fact that jitter contributes to I_{ijc}^{Set1}, whereas our improvement only affects $I_{ijc}^{\mathit{Set2}}(t)$. As I_{ijc}^{Set1} account for an increasingly larger fraction of the total response time, the relative improvement of (Tight) decreases. However, the absolute response-time improvement (not shown) and the number of improvements (bottommost graph) is not noticeably affected by the jitter. As the jitter grows larger than the period (or larger than several periods) the effects of our improvements diminish further. However, systems with such large jitter are rare in control-systems (which constitute the majority of real-time systems), where the jitter is typically only allowed to be a few percent of the period. Also, for system with such large jitters (such as multimedia applications), other methods [PG99, Red03] to reduce the estimated response time can be used.

Finally, figure 1.15 corresponds to a configuration where the number of transactions is 1, system load 80%, and load of the task under admission control is 2%. This type of scenario would occur in a system using a hybrid scheduling method, supporting both static cyclic scheduled tasks (corresponding to the single high priority transaction) and priority scheduled tasks running in the background of the static schedule [MTS02]. This situation shows where our method excels. All tasks have offset relations among them, resulting in well over 30% better admission probability (4–9 tasks) over (Orig) and an average improvement of over 50% when the number of tasks/transaction is more than 8. Another interesting thing is that (Exact) and (Tight) always yield exact response times. This comes from the fact that when considering a transaction in isolation (no interference among several transactions) the slanted stair interference function captures the worst case interference exactly.

1.6 Conclusions and Future Work

We have presented an improvement that calculates tighter (lower) response times than does earlier approximation methods. We prove that our method never calculates greater response times than the method in [PG98]. Furthermore we prove that our method never underestimates the interference caused by higher priority tasks. Hence, it calculates a safe and tight approximation of the actual worst-case response time.

We exploit a misconception in previous methods concerning the interference a task poses on a lower priority one. The concept “imposed” interference is introduced, and is shown to more accurately capture this interference compared to the previously accepted concept of “released for execution” interference. This situation is analogous to floating point addition where “released for execution” interference corresponds to calculations with integer values (rounded up) whereas “imposed” interference corresponds to calculations with the more accurate floating point values (resulting in a lower total sum).

Simulations show that the improvement is significant (especially when tasks/transaction ratio is high), typically about 15% tighter response times in 50% of the cases, resulting in 12% higher admission probability for low priority task subjected to admission control. In certain circumstances the improvement is much greater, and with just one transaction (corresponds to a static schedule) our proposed method calculates exact response times.

Our tighter analysis is noticeably slower than the original analysis (even slower than the exact for small task sets). This is a natural effect of using tighter interference functions since it gives slower fix-point convergence. However, we have previously proposed a method to speed up the original analysis [MTN04]. In our future work we will adapt that method to our tight analysis. We will also incorporate complementary improvements to RTA for tasks with offsets such as [PG99, Red03].

References

[ABD$^+$95] N.C. Audsley, A. Burns, R.I. Davis, K. Tindell, and A.J. Wellings. Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.

[ABT$^+$93] N.C. Audsley, A. Burns, K. Tindell, M.F. Richardson, and A.J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[JP86] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[LL73] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[MTN04] Jukka Mäki-Turja and Mikael Nolin. Faster Response Time Analysis of Tasks With Offsets. In *Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS)*, May 2004.

[MTS02] J. Mäki-Turja and M. Sjödin. Combining Dynamic and Static Scheduling in Hard Real-Time Systems. Technical Report MRTC no. 71, Mälardalen Real-Time Research Centre (MRTC), October 2002.

[PG98] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[PG99] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 328–339, December 1999.

[Red03] O. Redell. Accounting for Precedence Constraints in the Analysis of Tree-Shaped Transactions in Distributed Real-Time Systems. Technical Report TRITA-MMK 2003:4, Dept. of Machine Design, KTH, 2003.

[SH98] M. Sjödin and H. Hansson. Improved Response-Time Calculations. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998. URL: http://www.docs.uu.se/~mic/papers.html.

[Tin92] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

Appendix A: Complete RTA formulae

In this appendix we provide the complete set of formulae to calculate the worst case response time, R_{ua}, for a task under analysis, τ_{ua}, as presented in Palencia Gutiérrez *et al.* [PG98].

The interference transaction Γ_i poses on a lower priority task, τ_{ua}, if τ_{ic} coincides with the critical instant, is defined by (see equation 1.5 in this paper):

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(\left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor + \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil \right) C_{ij} \tag{26 in [PG98]}$$

where the phase between task τ_{ij} and the candidate critical instant task τ_{ic} is defined as (see equation 1.3 in this paper):

$$\Phi_{ijc} = T_i - (O_{ic} + J_{ic} - O_{ij}) \mod T_i \tag{17 in [PG98]}$$

The approximation function for transaction Γ_i which considers all candidate τ_{ic}-s simultaneously, is defined by (see equation 1.6 in this paper):

$$W_i^*(\tau_{ua}, w) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, w) \tag{27 in [PG98]}$$

The length of a busy period, for τ_{ua}, assuming τ_{uc} is the candidate critical instant, is defined as (Note that the approximation function is not used for Γ_u):

$$\begin{aligned} L_{uac} = & B_{ua} + (p_{L,uac} - p_{0,uac} + 1) C_{ua} + \\ & W_{uc}(\tau_{ua}, L_{uac}) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, L_{uac}) \end{aligned} \tag{30 in [PG98]}$$

where $p_{0,uac}$ denotes the first, and $p_{L,uac}$ the last, task instance, of τ_{ua}, activated within the busy period. They are defined as:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1 \tag{29 in [PG98]}$$

and

$$p_{L,uac} = \left\lceil \frac{L_{uac} - \Phi_{uac}}{T_u} \right\rceil \tag{31 in [PG98]}$$

In order to get the worst case response time for τ_{ua}, we need to check the response time for every instance, $p \in p_{0,uac} \dots p_{L,uac}$, in the busy period. Completion time of the p'th instance is given by:

$$\begin{aligned} w_{uac}(p) = & B_{ua} + (p - p_{0,uac} + 1)C_{ua} \\ & + W_{uc}(\tau_{ua}, w_{uac}(p)) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, w_{uac}(p)) \end{aligned} \quad \text{(28 in [PG98])}$$

The corresponding response time (for instance p) is then:

$$R_{uac}(p) = w_{uac}(p) - \Phi_{uac} - (p-1)T_u + O_{ua} \quad \text{(32 in [PG98])}$$

To obtain the worst case response time, R_{ua}, for τ_{ua}, we need to consider every candidate critical instant ,τ_{uc} (including τ_{ua} itself), and for each such candidate every possible instance, p, of τ_{ua}:

$$R_{ua} = \max_{\forall c \in hp_u(\tau_{ua}) \cup a} [\max_{p = p_{0,uac}, \dots, p_{L,uac}} (R_{uac}(p))] \quad \text{(33 in [PG98])}$$

CHAPTER 2

Paper B: Efficient Response-Time Analysis for Tasks with Offsets

JUKKA MÄKI-TURJA AND MIKAEL NOLIN

In the proceedings of the 10th IEEE Real-time Technology and Applications Symposium (RTAS).

TORONTO CANADA, MAY 2004.

Abstract

We present a method that enables an efficient implementation of the approximate response-time analysis (RTA) for tasks with offsets presented by Tindell [Tin92] and Palencia Gutiérrez *et al.* [PG98].

The method allows for significantly faster implementations of schedulability tools using RTA. Furthermore, reducing computation time, from tens of milliseconds to just a fraction of a millisecond, as we will show, is a step towards on-line RTA in for example admission control systems.

We formally prove that our reformulation of earlier presented equations is correct and allow us to statically represent parts of the equation, reducing the calculations during fix-point iteration. We show by simulations that the speed-up when using our method is substantial. When task sets grow beyond a trivial number of tasks and/or transactions a speed-up of more than 100 times (10 transactions and 10 tasks/transaction) compared to the original analysis can be obtained.

2.1 Introduction

A powerful and well established schedulability analysis technique is the *Response-Time Analysis* (RTA) [ABD$^+$95]. RTA is applicable to systems where tasks are scheduled in strict priority order which is the predominant scheduling technique used in real-time operating systems today. In this paper, we present a method that enables an efficient implementation of the approximate RTA for tasks with offsets presented by Tindell [Tin92] and Palencia Gutiérrez *et al.* [PG98].

RTA is a method to calculate worst-case response times for tasks in hard real-time systems. In essence RTA is used to perform a schedulability test, i.e., checking whether or not tasks in the system will satisfy their deadlines. Traditionally, industrial use of schedulability tests has been limited. However, with recent advancements in software development and synthesis tools, such as UML-based tools [IL, Rat, Tel], schedulability tests can be integrated in the normal workflow and tool-chains used by real-time engineers.

This kind of tools can be used, for instance, to perform automatic allocation of tasks to nodes in a distributed real-time system or to automatically derive task priorities (priority assignment) so that task deadlines are guaranteed to be met. To be able perform such allocation and/or assignment tasks, tools need to be able to perform schedulability tests. Typically, such automatic allocation/assignment methods are based on optimization or search techniques, during which numerous possible configurations are evaluated. (There can easily be tens or hundreds of thousands of possible configurations even for small systems.) For each configuration a schedulability test is performed in order to evaluate different solutions. Hence, schedulability tests must be fast in order to be suitable for such systems.

Dynamic real-time systems, with on-line admission control of real-time tasks, needs to be able to quickly evaluate whether a dynamically arriving task can be admitted to the system. In these cases the tolerance for delays in the scheduling analysis is even less than in the case of software engineering tools.

Accounting for offsets between tasks gives significantly tighter analysis results than using the traditional notion of a critical instant where all tasks in the system are considered to be released simultaneously [LL73]. Hence, tools for automatic configuration (as well as on-line schedulability tests) would benefit from using this extension; it becomes easier to find feasible configurations. In fact, many systems that will be deemed infeasible by RTA without offsets will be feasible when taking offsets into account. However, the price of taking offsets into account is increased execution time of the analysis. Existing methods for RTA with offsets have all been focused on modeling capabilities while ignoring issues of computational complexity, e.g., [PG98, PG99, Red03, Tin92].

The first RTA for tasks with offsets was presented by Tindell [Tin92]. He provided an exact algorithm for calculating response time for tasks with offsets. However, this algorithm becomes computationally intractable for anything but small task sets due to its exponential time complexity. In order to deal with this problem, Tindell also provided an approximation algorithm, polynomial in time, which gives pessimistic but safe (worst case response time is never underestimated) results. Later, Palencia Gutiérrez *et al.* [PG98] formalized, generalized and improved Tindell's work.

In this paper we present a method that enables an efficient implementation of the approximate offset analysis given by Tindell [Tin92] and Palencia Gutiérrez *et al.* [PG98]. The correctness of our method is formally proven by demonstrating algebraic equivalence with the original methods. The method significantly speeds up the calculation of response times, as we will show by simulations.

Paper Outline: In section 2.2 we revisit and restate the original offset RTA [PG98, Tin92]. In section 2.3 we present our new method. Section 2.4 presents evaluations of our method, and finally, section 2.5 concludes the paper and outlines future work.

2.2 Existing offset RTA

This section revisits the existing response-time analysis for tasks with offsets [PG98, Tin92] and illustrates the intuition behind the analysis and the formulae.

2.2.1 System model

The system model used is as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a (periodic) sequence of events with period T_i (for non-periodic events T_i denotes the minimum inter-arrival time between two consecutive events). The activating events are mutually independent, i.e., phasing between them is arbitrary. A transaction, Γ_i, contains $|\Gamma_i|$ tasks, and each task is activated (released for execution) when a time, *offset*, has elapsed after the arrival of the external event.

We use τ_{ij} to denote a task. The first subscript denotes which transaction the task belongs to, and the second subscript denotes the number of the task within the transaction. A task, τ_{ij}, is defined by a worst case execution time

(C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The system model is formally expressed as follows:

$$\begin{aligned}\Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\}\\ \Gamma_i &:= \langle\{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i\rangle\\ \tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij}\rangle\end{aligned}$$

There are no restrictions placed on offset, deadline or jitter, i.e., they are allowed to be both smaller or greater than the period. Parameters for an example transaction (Γ_i) with two tasks (τ_{ia}, τ_{ib}) is visualized in figure 2.1. The offset denotes the earliest release time of a task relative to the start of its transaction and jitter denotes the variability in the release of the task. (In figure 2.1 the jitter is not graphically visualized.)

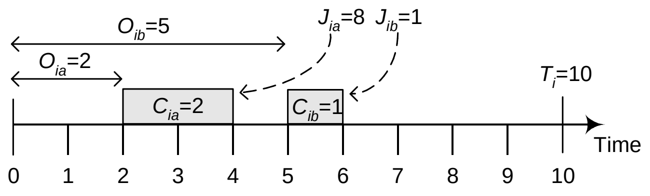


Figure 2.1: An example transaction Γ_i

2.2.2 Response-time analysis

The goal of RTA is to facilitate a schedulability test for each task in the system by calculating an upper bound on its worst case response time. We use τ_{ua} (task a, belonging to transaction Γ_u) to denote the *task under analysis*, i.e., the task who's response time we are currently calculating.

In the classical RTA (without offsets) the *critical instant* for τ_{ua} is when it is released at the same time as all higher (or equal) priority tasks [JP86, LL73]. In a task model with offsets this assumption yields pessimistic response times since some tasks can not be released simultaneously due to offset relations. Therefore, Tindell [Tin92] relaxed the notion of critical instant to be:

> At least one task in every transaction is to be released at the critical instant. (Only tasks with priority higher or equal to τ_{ua} are considered.)

Since it is not known which task that coincides with (is released at) the critical instant, every task in a transaction must be treated as a *candidate* to coincide with the critical instant.

Tindell's exact RTA tries every possible combination of candidates among all transactions in the system. This, however, becomes computationally intractable for anything but small task sets (the number of possible combinations of candidates is m^n for a system with n transactions and with m tasks per transaction). Therefore Tindell provided an approximate RTA that still gives good results but uses one single approximation function for each transaction. Palencia Gutiérrez *et al.* [PG98] formalized and generalized Tindells work. We will in this paper use the more general formalism of Palencia Gutiérrez *et al.*, although our proposed method is equally applicable to Tindell's original algorithm.

2.2.3 Interference function

Central to RTA is to capture the interference a higher or equal priority task (τ_{ij}) imposes on the task under analysis (τ_{ua}) during an interval of time t. Since a task can interfere with τ_{ua} multiple times during t we have to consider interference from possibly several *instances*. The interfering instances of τ_{ij} can be classified into two sets:

Set1 Activations that occur before or at the critical instant and that can be delayed by jitter so that they coincide with the critical instant.

Set2 Activations that occur after the critical instant

When studying the interference from an entire transaction Γ_i, we will consider each task, $\tau_{ic} \in \Gamma_i$, as a *candidate* for coinciding with the critical instant.

RTA of tasks with offsets is based on two fundamental theorems [PG98, Tin92]:

1. The worst case interference a task τ_{ij} imposes on τ_{ua} is when *Set1* activations are delayed by an amount of jitter such that they all occur at the critical instant and the activations in $Set2$ have zero jitter.

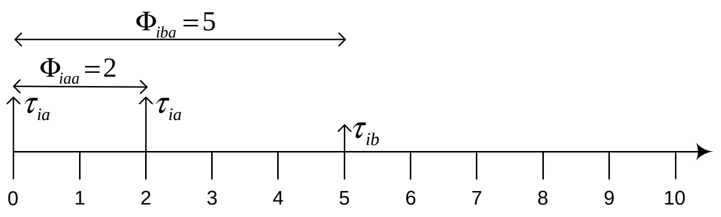
2. The task of Γ_i that coincide with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

The phasing between a task, τ_{ij}, and a critical instant candidate, τ_{ic}, becomes (slightly reformulated compared to [PG98], see Appendix 2.5):

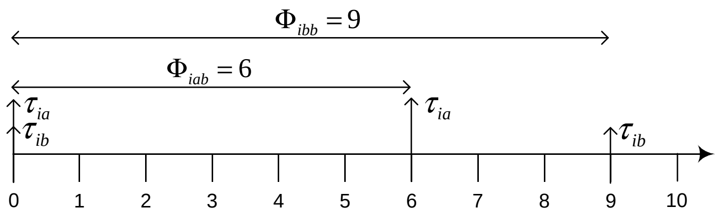
$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \mod T_i \tag{2.1}$$

From the second theorem we get that τ_{ic} will coincide with the critical instant after having experienced its worst case jitter delay (i.e., the critical instant will occur at $(O_{ic} + J_{ic}) \mod T_i$, relative to the start of Γ_i). This implies that the first instance of a task τ_{ij} in *Set2* will be released at Φ_{ijc} time units after the critical instant, and subsequent releases will occur periodically every T_i.

Figure 2.2 illustrates the four different Φ_{ijc}-s that are possible for our example transaction in figure 2.1. The upward arrows denote task releases (the height of the corresponding arrow denotes amount of execution released, i.e., C_{ia} and C_{ib} respectively). Figure 2.2(a) depicts the situation when τ_{ia} acts as the candidate critical instant. Shown is the phasing between τ_{ia} (2) and τ_{ib} (5) for this situation. Furthermore, figure 2.2(a) also shows activations for each task in the transaction. Task instances belonging to *Set1* are released at time 0, and the first instance belonging to *Set2* is also depicted (subsequent activation occur periodically). Figure 2.2(b) shows the corresponding situation if τ_{ib} happens to coincide with the critical instant.



(a) $\tau_{ic} = \tau_{ia}$



(b) $\tau_{ic} = \tau_{ib}$

Figure 2.2: Φ-s for the two candidates in Γ_i

Given the two sets of task instances (*Set1* and *Set2*) and the corresponding phase relative to the critical instant (Φ_{ijc}), the interference imposed by task τ_{ij}

can be divided into two parts:

1. the part imposed by instances in *Set1* (which is independent of time t), I^{Set1}_{ijc}, and
2. the part imposed by instances in *Set2* (which is a function of the considered time interval t), $I^{\mathit{Set2}}_{ijc}(t)$.

These are defined as follows:

$$I^{\mathit{Set1}}_{ijc} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij} \quad I^{\mathit{Set2}}_{ijc}(t) = \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij} \tag{2.2}$$

The interference transaction Γ_i poses on τ_{ua}, during a time interval t, when candidate τ_{ic} coincides with the critical instant, is:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I^{\mathit{Set1}}_{ijc} + I^{\mathit{Set2}}_{ijc}(t) \right) \tag{2.3}$$

Where $hp_i(\tau_{ua})$ denotes tasks belonging to transaction Γ_i, with priority higher or equal to the priority of τ_{ua}.

2.2.4 Approximation function

Since we beforehand cannot know which task in each transaction coincides with the critical instant, the exact analysis tries every possible combination [PG98, Tin92]. However, since this is computationally intractable for anything but small task sets the approximate analysis, presented in [PG98, Tin92], defines one single, upward approximated, function for the interference caused by transaction Γ_i:

$$W^*_i(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t) \tag{2.4}$$

That is, $W^*_i(\tau_{ua}, t)$ simply takes the maximum of each interference function (for each candidate τ_{ic}).

As an example consider again transaction Γ_i depicted in figure 2.1. Figure 2.3 shows the interference function for the two candidates (W_{ia} and W_{ib}), and it shows how W^*_i is derived from them by taking the maximum of the two functions at every t.

Given the interference (W^*_i) each transaction imposes on the task under analysis (τ_{ua}), during a time interval of length t, its response time (R_{ua}) can be calculated. Appendix 2.5 shows how to perform these response-time calculations.

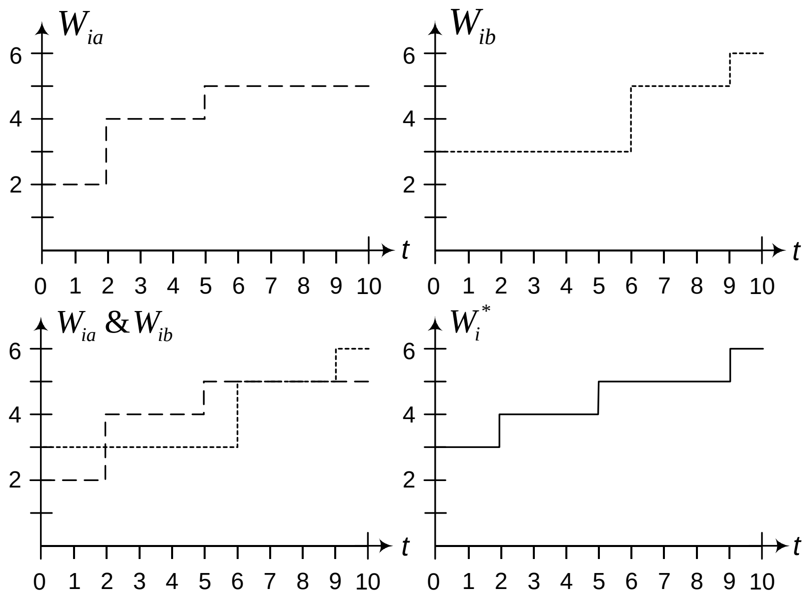


Figure 2.3: $W_{ic}(\tau_{ua}, t)$ and $W_i^*(\tau_{ua}, t)$ functions

2.3 Fast offset RTA

When calculating response times, the function $W_i^*(\tau_{ua}, t)$ (equation 2.4 on page 94) will be evaluated repeatedly. For each task and transaction pair (τ_{ua} and Γ_i) many different time-values, t, will be used during the fix-point calculations. However, since $W_i^*(\tau_{ua}, t)$ has a pattern that is repeated every T_i time units (see theorem 2.2 in this section), a lot of computational effort could be saved by representing the interference function statically, and during response-time calculation use a simple lookup function to obtain its value. This section shows how the function $W_i^*(\tau_{ua}, t)$ changes using such pre-computed information and how to calculate and store that information.

2.3.1 Approximation function with lookup

The key to make a static representation of $W_i^*(\tau_{ua}, t)$ is to recognisee that it contains two parts:

- A jitter induced part, denoted $J_i^{ind}(\tau_{ua})$. This part corresponds to the task instances belonging to *Set1*. Note that the amount of interference of these instances does not depend on t.
- A time induced part, denoted $T_i^{ind}(\tau_{ua}, t)$. This corresponds to task instances in *Set2*. The time induced part has a cyclic pattern that repeats itself every T_i units of time (as we will prove below).

We redefine equation 2.4 using our new notation as:

$$W_i^*(\tau_{ua}, t) = J_i^{ind}(\tau_{ua}) + T_i^{ind}(\tau_{ua}, t) \tag{2.5}$$

This partitioning of $W_i^*(\tau_{ua}, t)$ is visualized in figure 2.4. $J_i^{ind}(\tau_{ua})$ is the maximum starting value of each of the $W_{ic}(\tau_{ua}, t)$ functions (i.e. maximum of all $W_{ic}(\tau_{ua}, 0)$, see equation 2.3) which is calculated by:

$$J_i^{ind}(\tau_{ua}) = \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{\mathit{Set1}} \tag{2.6}$$

The time induced part, $T_i^{ind}(\tau_{ua}, t)$, represents the maximum interference, during t, from tasks activated after the critical instant, and is algebraically defined as:

$$T_i^{ind}(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua}, t) \tag{2.7}$$

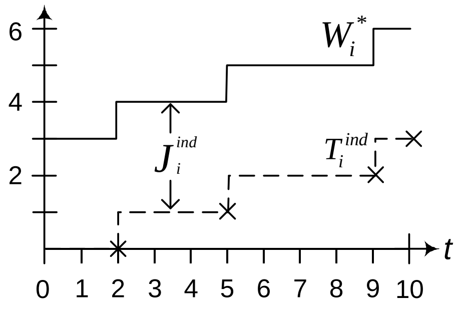


Figure 2.4: $W_i^*(\tau_{ua}, t)$, $J_i^{ind}(\tau_{ua})$, and $T_i^{ind}(\tau_{ua}, t)$

where

$$W_{ic}^{+}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t)\right) - J_i^{ind}(\tau_{ua}) \tag{2.8}$$

The correctness of our method requires that the definition of $W_i^*(\tau_{ua}, t)$ in equation 2.5 is functionally equivalent to the definition in equation 2.4.

THEOREM 2.1 *$W_i^*(\tau_{ua}, t)$ as defined in equation 2.4 and $W_i^*(\tau_{ua}, t)$ as defined in equation 2.5 are equivalent.*

PROOF REFERENCE. *The theorem is proved by algebraic equivalence in Appendix 2.5 .*

Further, in order to be able to make a static representation of $W_i^*(\tau_{ua}, t)$, we need to ensure that we store enough information to correctly reproduce $W_i^*(\tau_{ua}, t)$ for arbitrary large values of t. Since $T_i^{ind}(\tau_{ua}, t)$ is the only part of $W_i^*(\tau_{ua}, t)$ that is dependent on t, the following theorem gives that it is enough to store information for the first T_i time units:

THEOREM 2.2 *Assume $t = k * T_i + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_i$), then*

$$T_i^{ind}(\tau_{ua}, t) = k * T_i^{ind}(\tau_{ua}, T_i) + T_i^{ind}(\tau_{ua}, t')$$

PROOF REFERENCE. *The theorem is proved by algebraic equivalence in Appendix 2.5 .*

We represent $T_i^{ind}(\tau_{ua}, t)$ for the first T_i time units using the concave corners of the function $T_i^{ind}(\tau_{ua}, t)$ (marked with crosses in figure 2.4). The representation uses two arrays T_i^c and T_i^t. $T_i^c[x]$ represents the maximum amount of time induced interference Γ_i will pose on a lower priority task during interval lengths up to $T_i^t[x]$ ($x \in 1 \ldots |T_i^c|$). Using these two arrays we redefine $T_i^{ind}(\tau_{ua}, t)$ as follows:

$$\begin{aligned} T_i^{ind}(\tau_{ua}, t) =& k * T_i^c[|T_i^c|] + T_i^c[x] \\ k =& t \div T_i \\ t' =& t \text{ rem } T_i \\ x =& \min\{y \ : \ t' \leq T_i^t[y]\} \end{aligned} \tag{2.9}$$

For our example transaction, the time induced interference (represented in figure 2.4 by crosses) is stored in the arrays T_i^c and T_i^t as follows:

$$\begin{array}{ll} T_i^c & = [\quad 0, \quad 1, \quad 2, \quad 3] \\ T_i^t & = [\quad 2, \quad 5, \quad 9, \quad 10] \end{array}$$

Using equation 2.5 and equation 2.9 instead of equation 2.4 to compute the interference function $W_i^*(\tau_{ua}, t)$ will significantly reduce the time to compute response times as we will show in section 2.4.

2.3.2 Pre-computing T_i^c and T_i^t

To compute T_i^c and T_i^t we will first calculate the pattern for each $W_{ic}^+(\tau_{ua}, t)$ from which we will later extract the maximum. Hence, we have to consider each task τ_{ic} in Γ_i as a candidate to coincide with the critical instant. For each candidate task, τ_{ic}, we define a set of points p_{ic}. Each point $p_{ic}[k]$ has an x and a y coordinate, describing how the time induced interference grows over time if the corresponding τ_{ic} coincides with the critical instant. The points in p_{ic} corresponds to the convex corners of $W_{ic}^+(\tau_{ua}, t)$ of equation 2.8. W_{ia}^+ and W_{ib}^+, for our example transaction, are depicted in figure 2.5 and the corresponding p_{ia} and p_{ib} are illustrated by black and white circles respectively.

To calculate the set p_{ic}, we (without loss of generality) assume that tasks are enumerated according to their first activation after the critical instant, i.e.,

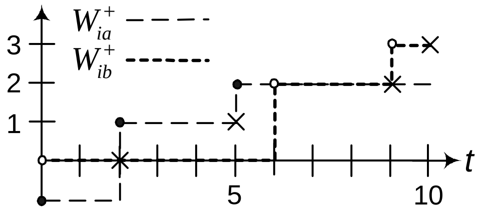


Figure 2.5: Visual representation of p_{ic} sets

according to Φ_{ijc} values. The following equations define the array p_{ic}:

$$p_{ic}[1].x = 0$$
$$p_{ic}[1].y = \sum_{\forall j \in hp_i(\tau_{ua})} I^{Set1}_{ijc} - J^{ind}_i(\tau_{ua})$$
$$k \in 2 \ldots |\Gamma_i| \begin{cases} p_{ic}[k].x = \Phi_{ikc} \\ p_{ic}[k].y = p_{ic}[k-1].y + C_{ik} \end{cases}$$

Each p_{ic} set represents how the time induced interference grows, for critical instant candidate τ_{ic}, during one period (T_i). For our example transaction of figure 2.1, we get the following two p_{ic}-s (corresponding to the black and white circles in figure 2.5):

$$\begin{array}{lll} p_{ia} & = [\langle 0, -1\rangle, \langle 2, 1\rangle, \langle 5, 2\rangle] & \text{black circles} \\ p_{ib} & = [\langle 0, \quad 0\rangle, \langle 6, 2\rangle, \langle 9, 3\rangle] & \text{white circles} \end{array}$$

Now, we have the information generated by all $W^+_{ic}(\tau_{ua}, t)$-functions, stored in the p_{ic}-sets. These stepwise functions are represented by one point per step. In order to get a representation of $T^{ind}_i(\tau_{ua}, t)$ in equation 2.7, we extract the points that represents the maximum of all $W^+_{ic}(\tau_{ua}, t)$-s. Thus, we will obtain the convex corners of $T^{ind}_i(\tau_{ua}, t)$.

Next, we calculate the set of points, p_i, as the union of all p_{ic}-s:

$$p_i = \bigcup_{\tau_{ic} \in \Gamma_i} p_{ic}$$

In order to determine what points in p_i that corresponds the the convex corners of $T^{ind}_i(\tau_{ua}, t)$, we define the relation *subsumes* that says: A point $p_i[a]$ subsumes a point $p_i[b]$ (denoted $p_i[a] \succ p_i[b]$) if the presence of $p_i[a]$

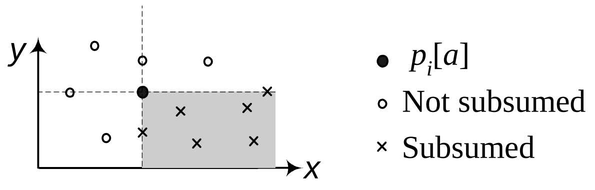


Figure 2.6: Removing points from p_i

implies that $p_i[b]$ is not a convex corner. Figure 2.6 illustrates the subsumes relation graphically, and the formal definition is:

$$p_i[a] \succ p_i[b] \text{ iff } p_i[a].y \geq p_i[b].y \wedge p_i[a].x \leq p_i[b].x$$

Given the subsumes relation the convex corner are found by removing all subsumed points:

$$\text{From } p_i \text{ remove } p_i[b] \text{ if } \exists a \neq b : p_i[a] \succ p_i[b] \tag{2.10}$$

Now, p_i contains the convex corners of the function $T_i^{ind}(\tau_{ua}, t)$. For our example transaction we now have:

$$p_i = [\langle 0,0\rangle, \langle 2,1\rangle, \langle 5,2\rangle, \langle 9,3\rangle]$$

All we have to do now is to find the concave corners (illustrated by crosses in figure 2.5) and store them in the arrays T_i^c and T_i^t. This is done by the following algorithm:

```
for k := 1 to $|p_i|$ do
    $T_i^c[k]$ := $p_i[k].y$
    if k < $|p_i|$ then
        $T_i^t[k]$ := $p_i[k+1].x$
    else
        $T_i^t[k]$ := $T_i$
done
```

For our example transaction this gives the following T_i^c and T_i^t (corresponding to crosses in figure 2.5):

$$\begin{array}{ll} T_i^c & = [\ 0,\ 1,\ 2,\ 3] \\ T_i^t & = [\ 2,\ 5,\ 9,\ 10] \end{array}$$

In the special case that some task τ_{ij} has $\Phi_{ijc} = 0$, the first element of T_i^c may not be zero. However, since $T_i^{ind}(0) = 0$, we need to have at least one element in T_i^c that is zero. In such cases we prepend both the arrays T_i^c and T_i^t with a zero (stating that there will be 0 time induced interference for any time interval of length up to 0).

2.3.3 Space and Time Complexity

The number of points to calculate (p_i) is quadratic with respect to the number of tasks in the transaction Γ_i ($|\Gamma_i|$ points for each candidate task). Thus, storing T_i^c and T_i^t results in a quadratic space complexity since, in the worst-case, no points from p_i will be removed.

The method presented in this paper divides the calculation of W_i^* into a pre-calculation and a fix-point iteration phase. A naive implementation of the removal procedure in equation 2.10 requires comparison of each pair of points; resulting in cubic time-complexity ($O(|\Gamma_i|^3)$) for pre-calculating T_i^c and T_i^t.[1] During the fix-point iteration phase, a binary search through a quadratically sized array is performed (equation 2.9), resulting in $O(\log |\Gamma_i|^2)$ time complexity for calculating W_i^* according to equation 2.5. The original complexity for calculating W_i^* according to equation 2.4 is $O(|\Gamma_i|^2)$.

In a complete comparison of complexity, the calculation of $W_i^*(\tau_{ua}, t)$ must be placed in its proper context (see the response-time formulae in appendix A). Assume X denotes number of fix-point iterations needed, then the overall complexity for the original approach (equation 2.4) is ($O(X|\Gamma_i|^2)$), whereas our method (equation 2.5 and equation 2.9) yields ($O(|\Gamma_i|^3 + X \log |\Gamma_i|^2)$).

2.4 Evaluation

In order to evaluate the effectiveness of our method we have implemented the response-time equations in appendix 2.5, using both the original definition of W_i^* from section 2.2 (Old RTA) and our faster version of W_i^* from section 2.3 (Fast RTA). Using these implementations and a synthetic task-generator we have performed an evaluation, by simulations, of both approaches by calculating the response times for all tasks in the system.

[1] In section 2.4 we use an $O(|\Gamma_i|^2 logN)$ implementation based on sorting the points and making a single pass through the sorted set.

2.4.1 Description of Simulation

In our simulator we generate task sets that are used as input to the different RTA implementations. The task-set generator takes the following parameters as input:

- Total system load (in % of total CPU utilization),
- the number of transactions to generate, and
- the number of tasks per transaction to generate.

Using these parameters a task set with the following properties is generated:

- The total system load is proportionally distributed over all transactions in the system.
- Transaction periods (T_i) are randomly distributed in the range 1.000 to 1.000.000 (uniform distribution).
- Each offset (O_{ij}) is randomly distributed within the transaction period (uniform distribution).
- The execution times (C_{ij}) are chosen as a fraction of the time between two consecutive offsets in the transaction. The fraction is the same throughout one transaction. The fraction is selected so that the the transaction load (as defined by the first property) is obtained.
- The jitter (J_{ij}) is randomly distributed between zero and 1.2 times the transaction period ($0..1.2T_i$, uniform distribution).
- Blocking (B_{ij}) is set to zero.
- The priorities are assigned in rate monotonic order [LL73].

We have measured execution times for performing RTA (for all tasks in the system) using both methods (Old RTA and Fast RTA). The execution times are obtained from a laptop with a Pentium III CPU. For Fast RTA the execution times include the time to calculate T_i^c and T_i^t. The results in section 2.4.2 have been obtained by taking the mean values of 50 simulated task-sets for each point in each graph. The 95% confidence intervals are shown for all execution times (although difficult to see due to their small size).

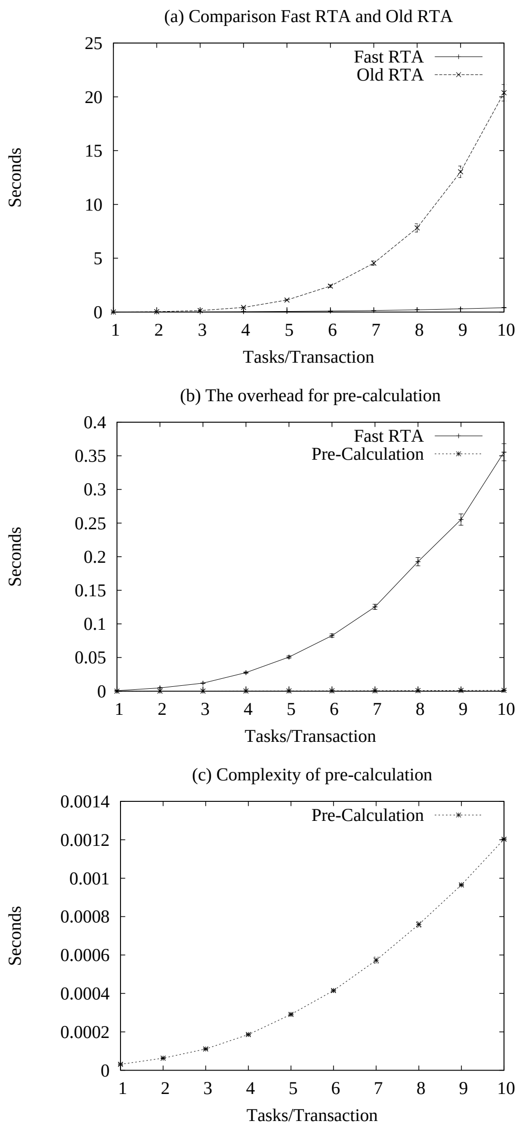


Figure 2.7: Execution time

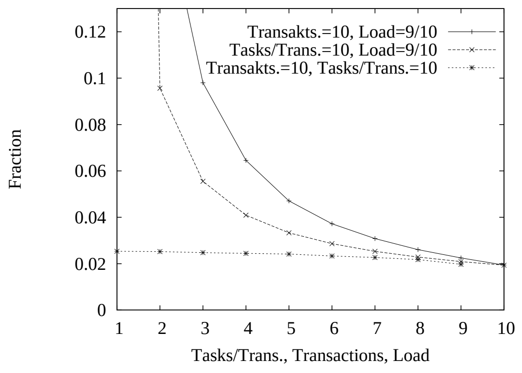


Figure 2.8: Relative execution time

2.4.2 Simulation Results

Figure 2.7(a) shows the execution times for Fast RTA and Old RTA when the number of tasks/transaction is varied from 1 to 10 (while keeping the system load at 9/10 (90%) and the number of transactions at 10). When the number of tasks/transaction is 10, the execution time is less than 0.40 seconds for Fast RTA, and about 20 seconds for Old RTA. This amounts to a speedup of 50 times. Similar execution times are obtained both when varying the number of transactions between 1 and 10 and when varying load between 1/10 (10%) and 9/10.

In figure 2.7(b) the complexity of Fast RTA is shown, and by comparison with figure 2.7(a) it can be seen that Fast RTA has a less steep curve than does Old RTA. Also, in figure 2.7(b) the amount of time spent pre-calculating the arrays T_i^c and T_i^t is plotted, and it is apparent that the overhead is negligible. For the larger task sets, about 0.3% of the total time of Fast RTA, is spent on pre-computing T_i^c and T_i^t.

Figure 2.7(c) shows the execution-time of the pre-calculation only. Since we use a $O(N^2 \log N)$ implementation of the pre-calculation the slope is slightly less than what could be expected from a naive implementation ($O(N^3)$).

In figure 2.8 we show the relative execution time of Fast RTA compared to Old RTA, calculated by t_{Fast}/t_{Old}, where t_{Fast} is the execution time for Fast RTA and t_{Old} for Old RTA. The first plot (+) shows how the relative execution time changes when the number of tasks/transaction is varied from 3 to 10.

When the number of tasks/transaction is 1 the relative execution time is 0.58 and it rapidly decreases to the values visible in the graph.

The second plot ($\times$) in figure 2.8 illustrates when the number of transactions is varied between 2 and 10. When the number of transactions is 1, the relative execution time is 1.01, which means that Fast RTA is *slower* than Old RTA. When performing RTA for a single transaction, the overhead of pre-computing T_i^c and T_i^t outweighs the benefits obtained during the RTA (the pre-computed W_i^* is never used). However, as seen in the plot, when the number of transactions is higher than 1, the overhead is well justified since the total RTA is significantly faster.

The third plot ($*$) in figure 2.8 illustrates when the load is varied between 1/10 (10%) and 9/10 (90%). In this plot we see that the relative execution time is not highly dependent on the system load, only a small decrease in relative execution time is obtained as the system load grows.

In order to compare Old RTA and Fast RTA, in the context of on-line admission control, we generated task sets with 9/10 load, 10 transactions with 10 tasks/transaction and performed the RTA for a single task (corresponding to a dynamically arriving task to the system) at lowest priority. We generated 100 different tasks sets using execution times for the single task between 1000 and 6000. The result was that the average execution time for Fast RTA was 0.33ms and 44ms for Old RTA. The speedup for admission control is about 130 times, which is noticeable greater than in the previous simulations. The reason is that the new task is the only task in its transaction, which means that W_i^* is used for all interference computations and W_{ic} is never used (see Appendix 2.5) and hence our improvement to W_i^* is isolated. In fact, in the preliminary work for this paper [MTN03] a speedup of over 600 times was observed for a simplified task-model where fix-point iteration only required W_i^* to be computed.

Our conclusions from this simulation study are that: (1) Fast RTA performs significantly better than Old RTA. For anything but trivially small task sets the speedup is at least in the order of a magnitude, (2) Fast RTA brings down execution times for whole scenarios from the order of seconds to fractions of seconds, and (3) Fast RTA brings down execution times for single tasks from the order of some 100ms to the microsecond range. This decrease is important in order to make RTA a feasible technique to include in, e.g., on-line scheduling algorithms performing RTA on-line (admission control being an example) and optimizing allocation or configuration tools.

2.5 Conclusions and Future Work

In this paper we have presented a novel method that allows for an efficient implementation of the approximate Response-Time Analysis (RTA) for tasks with offsets presented by Tindell [Tin92] and Palencia Gutiérrez *et al.* [PG98].

The main effort in performing RTA for tasks with offsets is to calculate how higher (or equal) priority tasks interfere with a task under analysis. The essence of our method is to calculate and store this information statically and during response-time calculations (fix-point iteration), use a simple table lookup. We have formally proved that the RTA-equations can be reformulated to allow such static representation of task interference.

We have, by simulations, shown that the speedup for our method compared to [PG98] is substantial. For realistically sized task sets (100 tasks), performing schedulability analysis gives a speedup of about 50 times. And from our evaluation we can conjecture that the relative improvement will be even higher for larger task sets. In an on-line RTA context, e.g., on-line admission control systems, our method outperforms previous methods by at over a factor of 100 and reducing the actual time to the micro second range.

Faster RTA have several positive practical implications: (1) Engineering tools (such as those for task allocation and priority assignment) can feasible rely on RTA and use the task model with offsets, and (2) on-line scheduling algorithms, e.g., those performing admission control, can use accurate on-line schedulability tests based on RTA.

We have earlier provided a tighter version of the RTA for tasks with offsets [MTS03]. Our next step is to extend our method of static representation of task interference to our tighter RTA, yielding a RTA that is both significantly faster and provides less pessimistic response times than previous techniques. Further, we are currently starting a project where RTA for tasks with offsets will be used in software engineering tools. The RTA will be used both to perform schedulability tests and for automatic allocation of software to nodes in a distributed system.

References

[ABD$^+$95] N.C. Audsley, A. Burns, R.I. Davis, K. Tindell, and A.J. Wellings. Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.

[IL] I-Logix. Rhapsody. http://www.ilogix.com/products/rhapsody.

[JP86] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[LL73] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[MTN03] Jukka Mäki-Turja and Mikael Nolin. Faster Response Time Analysis of Tasks With Offsets. In *24th IEEE Real-Time Systems Symposium (RTSS) Work In Progress Proc.*, December 2003.

[MTN04] Jukka Mäki-Turja and Mikael Nolin. Speeding Up the Response-Time Analysis of Tasks with Offsets. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-154/2004-1-SE, MRTC Report, Mälardalen Real-Time Research Centre, Mälardalen University, February 2004.

[MTS03] J. Mäki-Turja and M. Sjödin. Improved Analysis for Real-Time Tasks With Offsets – Advanced Model. Technical Report MRTC no. 101, Mälardalen Real-Time Research Centre (MRTC), May 2003.

[PG98] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[PG99] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 328–339, December 1999.

[Rat] Rational. Rational Rose RealTime. http://www.rational.com/-products/rosert.

[Red03] O. Redell. *Response time analysis for implementation of distributed control systems*. PhD thesis, KTH, Department of Machine Design, 2003. Series: TRITA-MMK 2003:17.

[Tel] TeleLogic. Telelogic tau. http://www.telelogic.com/products/tau.

[Tin92] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

Appendix A: Complete RTA formulae

In this appendix we provide the complete set of formulae to calculate the worst case response time, R_{ua}, for a task under analysis, τ_{ua}, as presented in Palencia Gutiérrez *et al.* [PG98].

The interference transaction Γ_i poses on a lower priority task, τ_{ua}, if τ_{ic} coincides with the critical instant, is defined by (see equation 2.3 in this paper):

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(\left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor + \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil \right) * C_{ij} \tag{26 in [PG98]}$$

where the phase between task τ_{ij} and the candidate critical instant task τ_{ic} is defined as (see equation 2.1 in this paper):

$$\Phi_{ijc} = T_i - (O_{ic} + J_{ic} - O_{ij}) \mod T_i \tag{17 in [PG98]}$$

The approximation function for transaction Γ_i which considers all candidate τ_{ic}-s simultaneously, is defined by (see equation 2.4 in this paper):

$$W_i^*(\tau_{ua}, w) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, w) \tag{27 in [PG98]}$$

The length of a busy period, for τ_{ua}, assuming τ_{uc} is the candidate critical instant, is defined as (Note that the approximation function is not used for Γ_u):

$$\begin{aligned} L_{uac} = & B_{ua} + (p_{L,uac} - p_{0,uac} + 1)C_{ua} + \\ & W_{uc}(\tau_{ua}, L_{uac}) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, L_{uac}) \end{aligned} \tag{30 in [PG98]}$$

where $p_{0,uac}$ denotes the first, and $p_{L,uac}$ the last, task instance, of τ_{ua}, activated within the busy period. They are defined as:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1 \tag{29 in [PG98]}$$

and

$$p_{L,uac} = \left\lceil \frac{L_{uac} - \Phi_{uac}}{T_u} \right\rceil \tag{31 in [PG98]}$$

In order to get the worst case response time for τ_{ua}, we need to check the response time for every instance, $p \in p_{0,uac} \ldots p_{L,uac}$, in the busy period. Completion time of the p'th instance is given by:

$$\begin{aligned} w_{uac}(p) =& B_{ua} + (p - p_{0,uac} + 1)C_{ua} \\ &+ W_{uc}(\tau_{ua}, w_{uac}(p)) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, w_{uac}(p)) \end{aligned} \quad \text{(28 in [PG98])}$$

The corresponding response time (for instance p) is then:

$$R_{uac}(p) = w_{uac}(p) - \Phi_{uac} - (p-1)T_u + O_{ua} \quad \text{(32 in [PG98])}$$

To obtain the worst case response time, R_{ua}, for τ_{ua}, we need to consider every candidate critical instant ,τ_{uc} (including τ_{ua} itself), and for each such candidate every possible instance, p, of τ_{ua}:

$$R_{ua} = \max_{\forall c \in hp_u(\tau_{ua}) \cup a} [\max_{p = p_{0,uac}, \ldots, p_{L,uac}} (R_{uac}(p))] \quad \text{(33 in [PG98])}$$

Appendix B: Proof of Theorems

In this appendix we provide proofs of theorems 2.1 and 2.2. We will perform all proofs by algebraic manipulation and use braces to highlight the expression that is manipulated in each step. We also annotate braces with the equations, properties, lemmas, or assumptions referred to when performing some manipulations. These proofs are also available in [MTN04].

When performing the manipulations we will, e.g., rely on the following properties:

(max) — The $\max_v$ operator allows terms that are constant with respect to the maximization variable (v) to be moved outside the maximization operation:

$$\max_v(X_v + Y) = \max_v(X_v) + Y.$$

(sum) — Summation over a set of terms can be divided into two separate summations:

$$\sum_v (X_v + Y_v) = \sum_v X_v + \sum_v Y_v$$

(ceil) — When taking the ceiling ($\lceil\ \rceil$) of a set of terms, terms that are known to be integers can be moved outside of the ceiling expression:

$$X \in \mathbb{N} \Rightarrow \lceil X + Y \rceil = X + \lceil Y \rceil$$

THEOREM 2.1 *$W_i^*(\tau_{ua}, t)$ as defined in equation 2.4 and $W_i^*(\tau_{ua}, t)$ as defined in equation 2.5 are equivalent.*

PROOF OF THEOREM 2.1

$$\underbrace{W_i^*(\tau_{ua}, t)}_{\textit{Eq.2.5}} = J_i^{ind}(\tau_{ua}) + \underbrace{T_i^{ind}(\tau_{ua}, t)}_{\textit{Eq.2.7}} =$$

$$J_i^{ind}(\tau_{ua}) + \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^+(\tau_{ua}, t)}_{\textit{Eq.2.8}} =$$

$$\underbrace{J_i^{ind}(\tau_{ua}) + \max_{\forall c \in hp_i(\tau_{ua})} \Bigg(\sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t) \big) - J_i^{ind}(\tau_{ua}) \Bigg)}_{\textit{(max)}} =$$

$$\underbrace{J_i^{ind}(\tau_{ua})} + \max_{\forall c \in hp_i(\tau_{ua})} \Bigg(\sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t) \big) \Bigg) - \underbrace{J_i^{ind}(\tau_{ua})} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t) \big)}_{\textit{Eq.2.3}} =$$

$$\underbrace{\max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t)}_{\textit{Eq.2.4}} = W_i^*(\tau_{ua}, t)$$

□

In proving theorem 2.2 we will use some lemmas.

LEMMA 2.1 *Regardless of candidate critical instant c: $I_{ijc}^{\textit{Set2}}(T_i) = C_{ij}$*

PROOF OF LEMMA 2.1

$$\underbrace{I_{ijc}^{\textit{Set2}}(T_i)}_{\textit{Eq.2.2}} = \underbrace{\left\lceil \frac{T_i - \Phi_{ijc}}{T_i} \right\rceil}_{0 \le \Phi_{ijc} < T_i \textit{ (Eq.2.1)}} C_{ij} = C_{ij}$$

□

LEMMA 2.2 *Assume* $t = k * T_i + t'$ *(where* $k \in \mathbb{N}$ *and* $0 \le t' < T_i$*), then* $I_{ijc}^{\mathit{Set2}}(t) = k * I_{ijc}^{\mathit{Set2}}(T_i) + I_{ijc}^{\mathit{Set2}}(t')$

PROOF OF LEMMA 2.2

$$\underbrace{I_{ijc}^{\mathit{Set2}}(t)}_{\mathit{Eq.2.2}} = \underbrace{\left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil}_{\mathit{Assumption}} C_{ij} = \underbrace{\left\lceil \frac{k * T_i + t' - \Phi_{ijc}}{T_i} \right\rceil}_{} C_{ij} =$$

$$\underbrace{\left\lceil \frac{k * T_i}{T_i} + \frac{t' - \Phi_{ijc}}{T_i} \right\rceil}_{\mathit{(ceil)} \wedge k \in \mathbb{N}} C_{ij} = \underbrace{\left(k + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil \right) C_{ij}}_{} =$$

$$k \underbrace{C_{ij}}_{\mathit{Lem.2.1}} + \underbrace{\left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij}}_{\mathit{Eq.2.2}} = k * I_{ijc}^{\mathit{Set2}}(T_i) + I_{ijc}^{\mathit{Set2}}(t')$$

□

LEMMA 2.3 $T_i^{ind}(\tau_{ua}, T_i) = \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}$

PROOF OF LEMMA 2.3

$$\underbrace{T_i^{ind}(\tau_{ua}, T_i)}_{\mathit{Eq.2.7}} = \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^{+}(\tau_{ua}, T_i)}_{\mathit{Eq.2.8}} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \Big(\underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\mathit{Set1}} + I_{ijc}^{\mathit{Set2}}(T_i) \right)}_{\mathit{(sum)}} - J_i^{ind}(\tau_{ua}) \Big) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \Big(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{\mathit{Set1}} + \sum_{\forall j \in hp_i(\tau_{ua})} \underbrace{I_{ijc}^{\mathit{Set2}}(T_i)}_{\mathit{Lem.2.1}} - J_i^{ind}(\tau_{ua}) \Big) =$$

$$\underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \Big(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{\mathit{Set1}} + \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} - J_i^{ind}(\tau_{ua}) \Big)}_{\mathit{(max)}} =$$

$$\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{\mathit{Set1}}}_{\mathit{Eq.2.6}} - J_i^{ind}(\tau_{ua}) =$$

$$\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{J_i^{ind}(\tau_{ua}) - J_i^{ind}(\tau_{ua})}_{} = \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}$$

□

THEOREM 2.2 *Assume $t = k * T_i + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_i$), then*

$$T_i^{ind}(\tau_{ua}, t) = k * T_i^{ind}(\tau_{ua}, T_i) + T_i^{ind}(\tau_{ua}, t')$$

PROOF OF THEOREM 2.2

$$\underbrace{T_i^{ind}(\tau_{ua},t)}_{\textit{Eq.2.7}} = \underbrace{W_{ic}^{+}(\tau_{ua},t)}_{\textit{Eq.2.8}} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + \underbrace{I_{ijc}^{\textit{Set2}}(t)}_{\textit{Lem.2.2}}\right) - J_i^{ind}(\tau_{ua}) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + k * \underbrace{I_{ijc}^{\textit{Set2}}(T_i)}_{\textit{Lem.2.1}} + I_{ijc}^{\textit{Set2}}(t')\right) -$$

$$J_i^{ind}(\tau_{ua}) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + kC_{ij} + I_{ijc}^{\textit{Set2}}(t')\right) - J_i^{ind}(\tau_{ua})}_{\textit{(sum)}} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})}$$

$$\underbrace{\left(\sum_{\forall j \in hp_i(\tau_{ua})} kC_{ij} + \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t')\right) - J_i^{ind}(\tau_{ua})\right)}_{\textit{(max)}} =$$

$$\underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} kC_{ij}} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t')\right) - J_i^{ind}(\tau_{ua}) =$$

$$k * \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}}_{\textit{Lem.2.3}} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t')\right) -$$

$$J_i^{ind}(\tau_{ua}) =$$

$$k * T_i^{ind}(\tau_{ua},T_i) +$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\textit{Set1}} + I_{ijc}^{\textit{Set2}}(t')\right) - J_i^{ind}(\tau_{ua})}_{\textit{Eq.2.8}} =$$

$$k * T_i^{ind}(\tau_{ua},T_i) + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua},t')}_{\textit{Eq.2.7}} =$$

$$k * T_i^{ind}(\tau_{ua},T_i) + T_i^{ind}(\tau_{ua},t')$$

□

CHAPTER 3

Paper C: Fast and Tight Response-Times for Tasks with Offsets

JUKKA MÄKI-TURJA AND MIKAEL NOLIN

To appear in the proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS).

PALMA DE MALLORCA SPAIN, JULY 2005.

Abstract

In previous work, we presented a tight approximate response-time analysis for tasks with offsets. While providing a tight bound on response times, the tight analysis exhibits similarly long execution times as does the traditional methods for calculating response-times for tasks with offsets. The existing method for fast analysis of tasks with offsets is not applicable to the tight analysis.

In this paper we extend the fast analysis to handle the distinguishing trait of the tight analysis; continuously increasing interference functions. Furthermore, we provide another speedup; by introducing pessimism in the modeling of interference at certain points, we speed up the convergence of the numerical solving for response-times without increasing the pessimism of the resulting response-times.

The presented fast-and-tight analysis is guaranteed to calculate the same response-times as the tight analysis, and in a simulation study we obtain speed-ups of more than two orders of magnitude for realistically sized tasks sets compared to the tight analysis. We also demonstrate that the fast-and-tight analysis has comparable execution time to that of the fast analysis. Hence, we conclude that the fast-and-tight analysis is the preferred analysis technique when tight estimates of response-times are needed, and that we do not need to sacrifice tightness for analysis speed; both are obtained with the fast-and-tight analysis.

3.1 Introduction

Response-Time Analysis (RTA) [ABD$^+$95] is a powerful and well established schedulability analysis technique. RTA is a method to calculate upper bounds on response-times for tasks in hard real-time systems. In essence RTA is used to perform a schedulability test, i.e., checking whether or not tasks in the system will satisfy their deadlines. RTA is applicable for, e.g., systems where tasks are scheduled in priority order which is the predominant scheduling technique used in real-time operating systems today.

Fast RTA has several practical implications, e.g., facilitating the use of response time calculations in an iterative workflow including automatic priority assignment and/or task allocation, or for admission control in on-line scheduling algorithms. Tighter response time allow for more efficient hardware utilization. Consequently, analysis speed and tight response time are desirable features in engineering resource constrained real-time systems.

To be able to calculate less pessimistic response times in systems where tasks may have dependencies in their release times, Tindell introduced RTA for a task model with offsets [Tin92]. Palencia and Harbour formalized and extended the work of Tindell in [PG98]. In [MTN04b] we have shown that the RTA for task with offset presented in their work calculates unnecessarily pessimistic response-times. As a remedy, we presented our tight analysis. The main source for this improvement comes from more accurate modeling of inter-task interference. In [PG98, Tin92] the interference only increases at discrete points in time, whereas in our tight analysis the interference can increase continuously over time. There is, however, a slight price to pay for this accuracy, slower fix-point convergence which can result in longer analysis time.

In this paper we extend our previous fast analysis for tasks with offsets [MTN04a] to enable its application to the tight analysis, providing a new method that calculates tight response times at fast analysis speed. The fast analysis has been shown to achieve two orders of magnitude speedup for realistically sized task sets [MTN04a]. The essence of this approach is to statically store the discrete points in time during the first period where the interference increases, and during equation solving use a simple and fast table lookup.

However, the approach taken in [MTN04a] is not directly applicable to the tight analysis since it uses a more accurate interference model where interference does not increase at discrete points in time. As a consequence, this introduces an additional problem; the interference does no longer exhibit a simple periodic pattern. Hence, the basic assumption of the fast analysis does not hold for the interference model of the tight analysis. One of the main contributions of this paper is to extend the fast analysis to cope with these traits of the tight analysis, enabling a fast-and-tight analysis.

Another main contribution is that we introduce, for the tight analysis, a method to speed up the numerical convergence during equation solving when calculating response-times. The method is based upon the insight that response-time equations cannot have solutions at arbitrary points in time (which we formally prove). At such points we modify the interference functions in such a way that numerical convergence is accelerated. Since the modifications are done only at times where no response-time solutions exist, they do not affect the final calculated response-time. Hence, the resulting analysis will calculate exactly the same response-times as does the tight analysis. This method is incorporated into the fast-and-tight analysis method.

Our third main contribution is a simulation study where we show that applying above methods to our tight method, the execution times of the resulting fast-and-tight analysis are comparable to those of the fast analysis. That is, we conclude that one does not have to sacrifice analysis speed to achieve accuracy, or vice versa, when using fast-and-tight analysis.

Paper Outline: Section 3.2 revisits our tight offset RTA [MTN04b]. In section 3.3 we present our tight and fast RTA. Section 3.4 presents an evaluation study, followed by conclusions in section 3.5.

3.2 Tight offset RTA

This section revisits our existing tight response-time analysis for tasks with offsets [MTN04b] and illustrates the intuition behind the analysis and the formulae.

3.2.1 System model

The system model used is as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i (For non-periodic events T_i denotes the minimum inter-arrival time between two consecutive events). The activating events are considered mutually independent, i.e., phasing between them are arbitrary. A transaction Γ_i contains $|\Gamma_i|$ number of tasks, and each task is activated (released for execution) when a relative time, *offset*, elapses after the arrival of the external event.

We use τ_{ij} to denote a task. The first subscript denotes which transaction the task belongs to, and the second subscript denotes the number of the task within the transaction. A task, τ_{ij}, is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The system model is formally expressed as:

$$\begin{aligned}\Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\}\\ \Gamma_i &:= \langle\{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i\rangle\\ \tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij}\rangle\end{aligned}$$

There are no restrictions placed on offset, deadline or jitter, e.g., they can each be either smaller or greater than the period. In [PG98] dynamic offsets are introduced, however they are modelled with the static offset and jitter parameters, and therefore the analysis technique presented here also straightforwardly applies to tasks with dynamic offsets. We assume that the load of the system, and each of the transactions, is less than 100%.[1]

Parameters for an example transaction (Γ_i) with two tasks (τ_{i1}, τ_{i2}) are depicted in figure 3.1. The offset denotes the earliest release time of a task relative to the start of its transaction and jitter (illustrated by the shaded region) denotes the variability in the release of the task. The upward arrows denote earliest possible release of a task and the size of the arrow corresponds to the released tasks execution time.

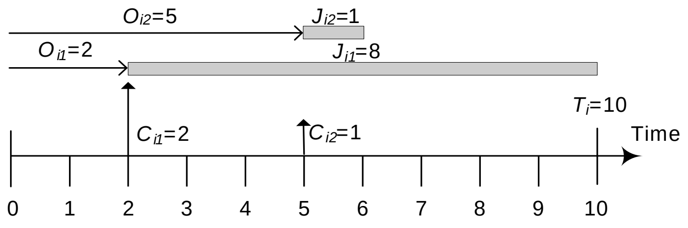


Figure 3.1: Example transaction

[1]This can easily be tested, and if not fulfilled some response-times may be infinite; rendering the system unschedulable.

3.2.2 Response-time analysis

The goal of RTA is to facilitate a schedulability test for each task in the system by calculating an upper bound on its worst case response-time. We use τ_{ua} (task a, belonging to transaction Γ_u) to denote the *task under analysis*, i.e., the task whose response time we are currently calculating.

In the classical RTA (without offsets) the *critical instant* for τ_{ua} is when it is released at the same time as all higher (or equal) priority tasks [JP86, LL73]. In a task model with offsets this assumption yields pessimistic response-times since some tasks can not be released simultaneously due to offset relations. Therefore, Tindell [Tin92] relaxed the notion of critical instant to be:

> At least one task in every transaction is to be released at the critical instant. (Only tasks with priority higher or equal to τ_{ua} are considered.)

Since it is not known which task coincides with (is released at) the critical instant, every task in a transaction must be treated as a *candidate* to coincide with the critical instant.

Tindell's exact RTA tries every possible combination of candidates among all transactions in the system. This, however, becomes computationally intractable for anything but small task sets. Therefore Tindell provided an approximate RTA that still gives good results but uses a single approximation function for each transaction. Palencia Gutierrez *et al.* [PG98] formalized and generalized Tindell's work.

3.2.3 Interference function

Central to RTA is to capture the interference a higher or equal priority task (τ_{ij}) causes the task under analysis (τ_{ua}) during an interval of time t (where $t = 0$ at the critical instant). Since a task can interfere with τ_{ua} multiple times during t we have to consider interference from possibly several *instances*. The interfering instances of τ_{ij} can be classified into two sets:

$Set1$ Activations that occur before or at the critical instant and that can be delayed by jitter so that they coincide with the critical instant.

$Set2$ Activations that occur after the critical instant

When studying the interference from an entire transaction Γ_i, we will consider each task, $\tau_{ic} \in \Gamma_i$, as a *candidate* for coinciding with the critical instant.

RTA for tasks with offsets is based on two fundamental theorems:

1. The worst case interference a task τ_{ij} causes τ_{ua} is when $Set1$ activations are delayed by an amount of jitter such that they all occur at the critical instant and the activations in $Set2$ have zero jitter.

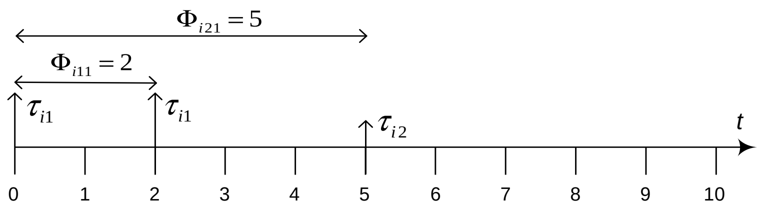
2. The task of Γ_i that coincide with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

The phasing between a task, τ_{ij}, and a critical instant candidate, τ_{ic}, becomes:

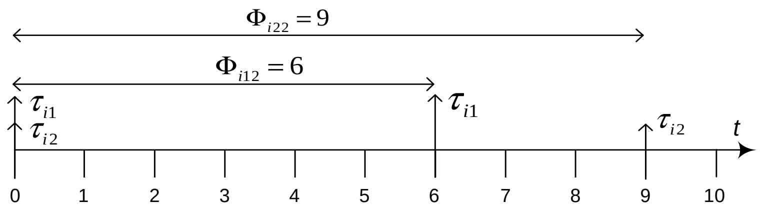
$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \mod T_i \tag{3.1}$$

This definition implies that the first instance of a task τ_{ij} in $Set2$ will be released at time $t = \Phi_{ijc}$, and subsequent releases will occur periodically every T_i.

Figure 3.2 illustrates the four different Φ_{ijc}-s that are possible for our example transaction of figure 3.1. The upward arrows denote task releases (the height of the corresponding arrow denotes amount of execution released, i.e., C_{i1} or C_{i2} respectively). Figure 3.2(a) the case that τ_{i1} coincides with the critical instant, where the phasing to τ_{i1} is 2 and to τ_{i2} is 5. Figure 3.2(b) shows the corresponding situation when τ_{i2} is the candidate to coincide with the critical instant.



(a) $\tau_{ic} = \tau_{i1}$



(b) $\tau_{ic} = \tau_{i2}$

Figure 3.2: Φ-s for the two candidates in Γ_i

Given the two sets of task instances ($Set1$ and $Set2$) and the corresponding phase relative to the critical instant (Φ_{ijc}), the worst-case interference during a time-interval t caused by task τ_{ij} can be divided into two parts:

1. The part caused by instances in $Set1$ (which is independent of the time interval t), I_{ijc}^{Set1}.
2. The part caused by instances in $Set2$ (which is a function of the time interval t), $I_{ijc}^{\mathit{Set2}}(t)$.

These are defined as follows:

$$\begin{aligned}
I_{ijc}^{\mathit{Set1}} &= \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij} \\
I_{ijc}^{\mathit{Set2}}(t) &= \left\lceil \frac{t^*}{T_i} \right\rceil C_{ij} - x \\
t^* &= t - \Phi_{ijc} \\
x &= \begin{cases} C_{ij} - (t^* \mod T_i) & \text{if } t^* > 0 \wedge \left(0 < t^* \mod T_i < C_{ij}\right) \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{3.2}$$

Note that, $I_{ijc}^{\mathit{Set2}}(t)$ is redefined compared to [PG98], resulting in lower (but still safe) response times. For more details and correctness proofs see [MTN04b].

The total interference transaction Γ_i imposes on τ_{ua}, during a time interval t, when candidate τ_{ic} coincides with the critical instant, is:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{\mathit{Set1}} + I_{ijc}^{\mathit{Set2}}(t)\right) \tag{3.3}$$

Where $hp_i(\tau_{ua})$ denotes tasks belonging to transaction Γ_i, with priority higher or equal to the priority of τ_{ua}.

3.2.4 Approximation function

Since we beforehand cannot know which task in each transaction coincides with the critical instant, the exact analysis tries every possible combination [Tin92, PG98]. However, since this is computationally intractable for anything but very small task sets the approximate analysis defines one single, upward approximated, function for the interference caused by transaction Γ_i:

$$W_i^*(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t) \tag{3.4}$$

$W_i^*(\tau_{ua}, t)$ simply takes the maximum of each interference function (one for each candidate τ_{ic}). As an example consider again transaction Γ_i depicted in figure 3.1. Figure 3.3 shows the interference function for the two candidates (W_{i1} and W_{i2}), and it shows how W_i^* is derived from them by taking the maximum of the two functions at every t.

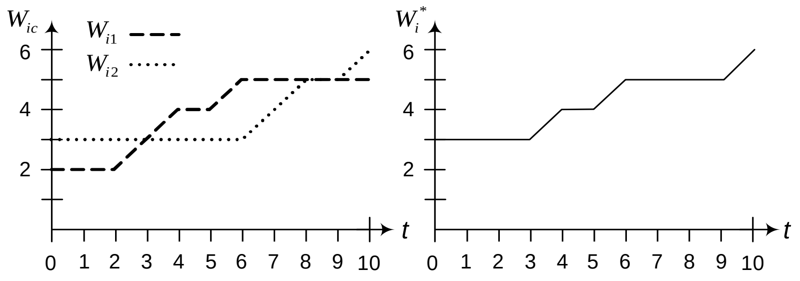


Figure 3.3: $W_{ic}(\tau_{ua}, t)$ and $W_i^*(\tau_{ua}, t)$ functions for example transaction

Given the interference (W_i^*) each transaction causes the task under analysis (τ_{ua}), during a time interval of length t, its response time (R_{ua}) can be calculated. The complete response time formulas provided by [PG98] can also be found in appendix 3.5.

3.3 Fast and Tight Analysis

When calculating response times, the function $W_i^*(\tau_{ua}, t)$ in equation 3.4 will be evaluated repeatedly. For each task and transaction pair (τ_{ua} and Γ_i) many different time-values, t, will be used during the fix-point calculations. For the traditional response-times analysis for tasks with offsets, a repetitive and periodic pattern for $W_i^*(\tau_{ua}, t)$ can easily be found, and a lot of computational effort is saved by representing the interference function statically, and during response-time calculations using a simple lookup function to obtain its value [MTN04a].

However, since the tight analysis deals with continuously increasing interference functions which do not exhibit a simple periodic pattern, the framework of [MTN04a] is not directly applicable to the tight analysis. This section shows how to find, calculate and store the periodic interference information for the tight RTA method. We also present how the function $W_i^*(\tau_{ua}, t)$ changes using such pre-computed information.

Furthermore, the continuous nature of interference in the tight analysis gives the tight analysis a computational disadvantage compared to the original analysis [PG98, Tin92]. In this section we will show how to remove this computational disadvantage by replacing the continuous interference functions with discretely increasing functions without introducing any pessimism in resulting response times.

3.3.1 The Periodicity of the Interference

The fundamental pre-requisite to statically represent the interference for a transaction, is that a repetitive pattern can be found (such that it suffices to store that pattern and use it to calculate the amount of interference for any time interval t). In our previous fast analysis [MTN04a], the full interference of each task within the transaction occurs within the first period (each task is released exactly once during each period). Hence, we could straight-forwardly represent the interference during the first period and reuse it for later periods.

However, in the tight analysis, the imposed interference of a task released towards the end of the period may not be fully included within the period. Even though the task is released within the period, the slanted interference function causes some of the interference to occur in the subsequent period. Figure 3.4 shows an example critical instant candidate where the interference from task z *spills* into next period.

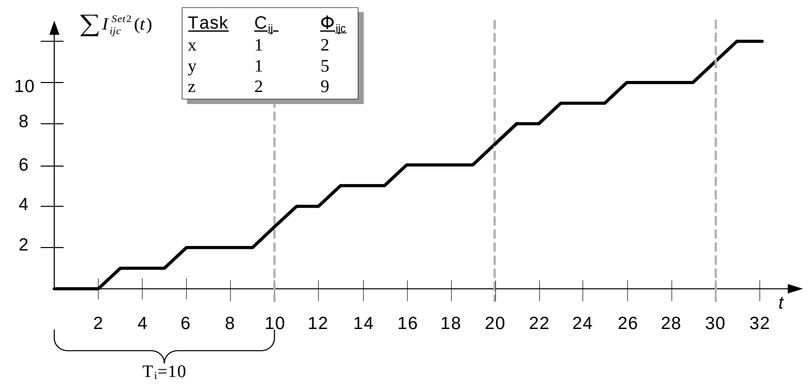


Figure 3.4: Interference spilling into the next period

As seen in figure 3.4, the interference for the first period differs from that of later periods. Obviously, there can be no spill during the first period, since tasks arriving before the critical instant (i.e. when $t < 0$) are accounted for in I_{ijc}^{Set1}. For subsequent periods, however, the effect of a task spilling over period boundaries will be identical. This means that for $t > T_i$ the interference is repetitive (with period = T_i) and allows for a static representation. The consequence of this is that we have to represent the interference for the first and subsequent periods separately.

3.3.2 Preliminaries

To prepare for subsequent calculations, we define three operations (order, merge, and split) that will be performed for each critical instant candidate before we proceed with calculation of a transactions' interference pattern. These transformations will not change the load or the timingbehavior of the interference, they only help us to restructure the information within a transaction.

Operation: Order Tasks are enumerated according to their first activation after the critical instant, i.e., according to increasing Φ_{ijc} values.

Operation: Merge Each task j' that is released before a previous task j has a chance to finish its execution, i.e. $(\Phi_{ijc} + C_{ij}) \mod T_i \geq \Phi_{ij'c}$, are merged into one task with execution time $C_{ij} + C_{ij'}$ and offset of Φ_{ijc}. This operation is performed until all possible tasks have been merged (and since the load of a transaction is less than 100% the process is guaranteed to converge).

Operation: Split When splitting a task, we define *spill* of a task j, belonging to transaction Γ_i for the critical instant candidate task c ($c \in \Gamma_i$), denoted S_{ijc}, as the amount of execution time that "spills over" into the next period. Since task j is released at time Φ_{ijc}, the amount of spill is:

$$S_{ijc} = \begin{cases} 0 & \text{if } \Phi_{ijc} + C_{ij} \leq T_i \\ \Phi_{ijc} + C_{ij} - T_i & \text{otherwise} \end{cases} \tag{3.5}$$

To make the spill explicit, we split each task j with a positive spill into 2 new tasks, denoted j' and j''. j' represents the amount of interference of task j that occurs within and at the end of the current period. j'' is called a *spill task* and represents the amount of interference that occurs at the beginning of the subsequent period. The definitions are:

$$\begin{array}{rlrl} C_{ij'} = & C_{ij} - S_{ijc} & \qquad C_{ij''} = & S_{ijc} \\ \Phi_{ij'c} = & \Phi_{ijc} & \qquad \Phi_{ij''c} = & 0 \end{array} \tag{3.6}$$

3.3.3 Jitter and time induced interference

The key to make a static representation of $W_i^*(\tau_{ua}, t)$ is to recognisee that it contains two parts:

- A jitter induced part, denoted $J_i^{ind}(\tau_{ua})$. This part corresponds to task instances belonging to $Set1$. Note that this interference is not dependent on t.
- A time induced part, denoted $T_i^{ind}(\tau_{ua}, t)$. This corresponds to task instances of $Set2$. With exception for the first period, the time induced part has a cyclic pattern that repeats itself every T_i (as proved below).

We redefine equation 3.4 using our new notation as:

$$W_i^*(\tau_{ua}, t) = J_i^{ind}(\tau_{ua}) + T_i^{ind}(\tau_{ua}, t) \tag{3.7}$$

This partitioning of $W_i^*(\tau_{ua}, t)$ is visualized in figure 3.5. $J_i^{ind}(\tau_{ua})$ is the maximum starting value of each of the $W_{ic}(\tau_{ua}, t)$ functions (i.e. max of $W_{ic}(\tau_{ua}, 0)$, see equation 3.3) which is calculated by:

$$J_i^{ind}(\tau_{ua}) = \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} \tag{3.8}$$

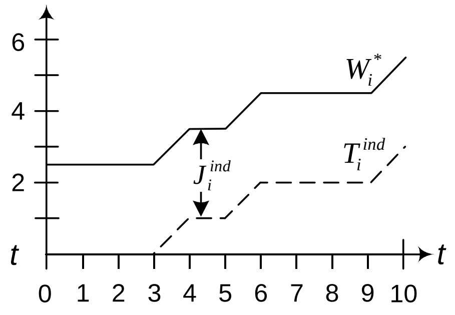


Figure 3.5: $W_i^*(\tau_{ua}, t)$, $J_i^{ind}(\tau_{ua})$, and $T_i^{ind}(\tau_{ua}, t)$

The time induced part, $T_i^{ind}(\tau_{ua}, t)$, represents the maximum interference, during t, from tasks activated after the critical instant. $T_i^{ind}(\tau_{ua}, t)$ is algebraically defined as:

$$T_i^{ind}(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^+(\tau_{ua}, t) \tag{3.9}$$

where

$$W_{ic}^{+}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t) \right) - J_i^{ind}(\tau_{ua}) \qquad (3.10)$$

Correctness criteria for our method requires that our new definition of $W_i^*(\tau_{ua}, t)$ in equation 3.7 is functionally equivalent to the definition in equation 3.4.

THEOREM 3.1 *$W_i^*(\tau_{ua}, t)$ as defined in equation 3.4 and $W_i^*(\tau_{ua}, t)$ as defined in equation 3.7 are equivalent.*

PROOF REFERENCE. *Proved by syntactic equivalence to Theorem 2.1 and corresponding proof.*

Further, in order to be able to make a static representation of $W_i^*(\tau_{ua}, t)$, we need to ensure that we store enough information to correctly reproduce $W_i^*(\tau_{ua}, t)$ for arbitrary large values of t. Since $T_i^{ind}(\tau_{ua}, t)$ is the only part of $W_i^*(\tau_{ua}, t)$ that is dependent on t, the following theorem gives that a periodicity of T_i exists in the interference:

THEOREM 3.2 *Assume spill tasks are accounted for, and $t = k * T_i + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_i$), then*

$$T_i^{ind}(\tau_{ua}, t) = k * T_i^{ind}(\tau_{ua}, T_i) + T_i^{ind}(\tau_{ua}, t')$$

PROOF REFERENCE. *The theorem is proved by algebraic equivalence in Appendix B.*

3.3.4 Representing time induced interference

In this section we show how the interference pattern of $T_i^{ind}(\tau_{ua}, t)$ can be calculated and represented statically. Since the first period should not account for any spill task, but subsequent periods should, we divide the presentation into two cases, one where spill task are not accounted for and one case where they are.

Spill task not accounted for

For each critical instant candidate, τ_{ic}, tasks are ordered, merged, and split according to section 3.3.2. Spill tasks are removed. We define a set of points p_{ic}, where each point $p_{ic}[k]$ has an x (representing time) and a y (representing interference) coordinate, describing how the time induced interference grows over time when τ_{ic} acts as the critical instant candidate. The points in p_{ic} correspond to the convex corners of $W^+_{ic}(\tau_{ua}, t)$ of equation 3.10. The following equations define the array p_{ic}:

$$\begin{aligned} p_{ic}[1].x &= 0 \\ p_{ic}[1].y &= \sum_{\forall j \in hp_i(\tau_{ua})} I^{\mathit{Set1}}_{ijc} - J^{ind}_i(\tau_{ua}) \\ p_{ic}[k].x &= \Phi_{ikc} + C_{ik} \qquad k \in 2 \dots |\Gamma_i| \\ p_{ic}[k].y &= p_{ic}[k-1].y + C_{ik} \quad k \in 2 \dots |\Gamma_i| \end{aligned} \tag{3.11}$$

$p_{ic}[1].y$ gives the initial relation (i.e. vertical distance at time 0) between different critical instant candidates, and is given by the difference in jitter-induced interference. Furthermore, the time-induced interference should be zero at time zero (illustrated in figure 3.5) which is achieved by subtracting the maximum of all jitter-induced interference (stored in $J^{ind}_i(\tau_{ua})$) when initializing $p_{ic}[1].y$ in equation 3.11.

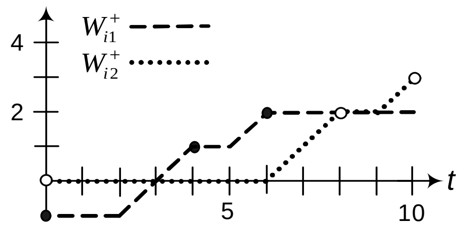


Figure 3.6: Visual representation of p_{ic} sets

The W^+_{i1} and W^+_{i2}, for our example transaction, are depicted in figure 3.6 and the corresponding p_{i1} and p_{i2} sets are illustrated by black and white circles respectively. For this example transaction we get the following two p_{ic}-s:

$$\begin{aligned} p_{i1} &= [\langle 0, -1\rangle, \langle 4, 1\rangle, \langle 6, 2\rangle] && \text{black circles} \\ p_{i2} &= [\langle 0, \quad 0\rangle, \langle 8, 2\rangle, \langle 10, 3\rangle] && \text{white circles} \end{aligned}$$

Now, the information generated by all $W_{ic}^{+}(\tau_{ua}, t)$-functions is stored in the p_{ic}-sets. To obtain the convex corners of $T_i^{ind}(\tau_{ua}, t)$, we need to extract the points that represent the maximum of all $W_{ic}^{+}(\tau_{ua}, t)$-s. To this end, we calculate the set of points, p_i, as the union of all p_{ic}-s:

$$p_i = \bigcup_{\tau_{ic} \in \Gamma_i} p_{ic}$$

In order to determine the points in p_i corresponding to the convex corners of $T_i^{ind}(\tau_{ua}, t)$, we define a *subsumes* relation: A point $p_i[a]$ subsumes a point $p_i[b]$ (denoted $p_i[a] \succ p_i[b]$) if the presence of $p_i[a]$ implies that $p_i[b]$ is not a convex corner. Figure 3.7 illustrates this relation graphically with a shaded region, and the formal definition is:

$$\begin{aligned}
&p_i[a] \succ p_i[b] \text{ iff}\\
&p_i[a].y \geq p_i[b].y \wedge \big(p_i[a].x - p_i[a].y \leq p_i[b].x - p_i[b].y\big)
\end{aligned}$$

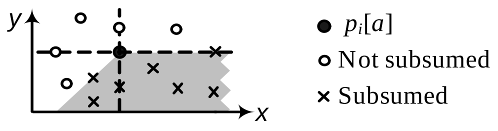


Figure 3.7: The subsumes relation

Given the subsumes relation, the convex corners are found by removing all subsumed points:

$$\text{From } p_i \text{ remove } p_i[b] \text{ if } \exists a \neq b : p_i[a] \succ p_i[b]$$

For our example transaction of figure 3.1 we have:

$$p_i = [\langle 0, 0\rangle, \langle 4, 1\rangle, \langle 6, 2\rangle, \langle 10, 3\rangle]$$

Spill task accounted for

Computing the set of points when accounting for spill tasks, denoted p_i', is analogous to computing p_i, with the following differences:

- Spill tasks from the split operation are not removed. Note that including a spill task might require an additional merge and order operation.

- In equation 3.11 on page 130 $p_{ic}[1].y$ defines the initial relation (difference in I_{ijc}^{Set1}) between different critical instant candidates. Since p'_i represents the time induced interference, $T_i^{ind}(\tau_{ua}, t)$, for $t \geq T_i$, $p'_{ic}[1].y$ should reflect this relation at the end of the first period. The interference for a critical instant c at the end of the first period is represented by $p_{ic}[|\Gamma_i|].y$, consequently we get the following modification to equation 3.11:[2]

$$p'_{ic}[1].y = p_{ic}[|\Gamma_i|].y - \max_{x \in \Gamma_i} p_{ix}[|\Gamma_i|].y$$

3.3.5 Increasing performance by removing slants

Assume that a set of points p_i (with or without spill tasks) has been calculated, representing the convex corners of the time induced interference function $T_i^{ind}(\tau_{ua}, t)$ during one period T_i. The points for our example transaction is illustrated in figure 3.8. Note that in the absence of spill tasks, the sets p_i and p'_i are identical.

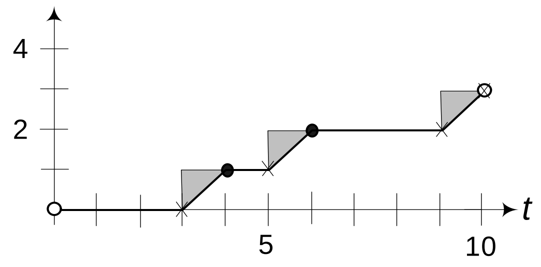


Figure 3.8: Remaining points and removal of slants

It can be proven that the fix-point iterative solution to Eq. 28 in [PG98] (see Appendix A), which is the equation where the interference function is used, cannot have any solution during the slants.

THEOREM 3.3 *Equation 28 in [PG98] cannot have a solution at a time t where any approximate interference function has a derivative greater than or equal to one.*

PROOF REFERENCE. *The theorem is proved in Appendix C.*

[2]Analogous to equation 3.11, we normalize the points to start at 0, hence we subtract the maximum of all $p_{ix}[|\Gamma_i|].y$.

No solutions to the response-time equation can exist during the slant of any interference function. Furthermore, the closest possible solution will be when the derivative of the interference function becomes zero. Hence, we can remove the slants and replace them with a stepped stair function, as illustrated by the grey areas of figure 3.8, without introducing any pessimism in the resulting response times. However, progress in the fix-point iteration is proportionally increased with any overestimation of the interference. Hence, by adding overestimation in the grey areas of figure 3.8 we will speed up the fix-point convergence without modifying the calculated response-times.

We will remove the slants by transforming the convex corners to concave corners (illustrated by crosses in figure 3.8[3]). The rules for finding the concave corners, v_i, from a set of convex corners, p_i, is as follows:

$$\begin{aligned} v_i[k].y =& p_i[1].y \\ v_i[k].x =& \begin{cases} p_i[k+1].x - (p_i[k+1].y - p_i[k].y) & \text{if } k < |p_i| \\ p_i[k].x & \text{if } k = |p_i| \end{cases} \\ k \in& 1 \ldots |p_i| \end{aligned}$$

The interpretation of v_i is as follows: For $t \leq T_i$, $v_i[k].y$ represents the maximum amount of time induced interference Γ_i will impose on a lower priority task during interval lengths up to $v_i[k].x$ ($k \in 1 \ldots |v_i|$). For our example transaction of figure 3.1, v_i becomes (indicated by crosses in figure 3.8 on the preceding page):

$$v_i = [\langle 3, 0\rangle, \langle 5, 1\rangle, \langle 9, 2\rangle, \langle 10, 3\rangle]$$

Note, especially that the final point (denoted $v_i[|v_i|]$) contains the sum of all interference during the period T_i.

In the special case that some task τ_{ij} has $\Phi_{ijc} = 0$ (e.g. in the case for spill tasks), $v_i[1].x$ will not be zero. However, since $T_i^{ind}(0) = 0$ (follows from equation 3.9), the first element of v_i needs to have x-value that is zero. In such cases we add the point $\langle 0, 0\rangle$ to v_i (stating that there will be 0 time induced interference for any time interval of length up to 0).

[3]While the last point in the set does not strictly represent a concave corner, it is still necessary for us to keep track of the amount of interference at the end of the period, hence that point will be included among the concave corners and is thus marked with a cross in the figure.

Discussion: Removal of slants

By removing the slants, we essentially revert to the stepped-stair interference functions used in the original analysis [PG98, Tin92]. This could seem surprising, since the tight analysis is based on the insight that stepped-stair interference functions are overly pessimistic. However, as theorem 3.3 states, there could be no response-time solutions during a slant. Hence, using slants *during fix-point equation solving* does not increase the precision of the analysis.[4]

However, when *deriving* the interference function it is imperative to use a faithful model (using slants) for the different sources of interference. Hence, once we have derived the interference function (as done when creating the point set p_i), we no longer need to represent the slants and can revert to a stepped-stair interference function.

An analogy could be made to calculations using floating-point values. If rounding values up before each calculation step, the resulting error will be greater than if the calculation is done using floating-point values, and only the final result is rounded up.

3.3.6 $T_i^{ind}(\tau_{ua}, t)$ using lookup

Since we need to represent the interference for the two first periods separately we will calculate the two point sets p_i (first period) and p_i' (second period) according to section 3.3.4. Next we will remove the slants for both these point sets as described in section 3.3.5 and store the new points in v_i and v_i' respectively.

Using the point sets v_i and v_i' we can calculate the interference from Γ_i for an arbitrary time t. For the first period the interference in v_i is used, and when $t > T_i$ we will start using the interference in v_i'. Using these point sets $T_i^{ind}(\tau_{ua}, t)$ can be reduced to fast lookup function, as follows:

[4]This is why the original response-time analysis [JP86] and exact analysis for tasks with offsets [PG98, Tin92] does not overestimate response times.

$$
\begin{aligned}
T_i^{ind}(\tau_{ua}, t) = &\begin{cases} v[n].y & \text{if } k < 1 \\ V & \text{if } k \geq 1 \end{cases} \\
V =& v_i[|v_i|].y + (k-1) * v_i'[|v_i'|].y + v_i'[n'].y \\
k =& t \div T_i \\
t' =& t \text{ rem } T_i \\
n =& \min\{m \ : \ t' \leq v_i[m].x\} \\
n' =& \min\{m \ : \ t' \leq v_i'[m].x\}
\end{aligned}
\tag{3.12}
$$

where k represents the number of whole periods (T_i) in t, and t' is the part of t that extends into the final period. It could be noted that $v_i[|v_i|].y$ contains the sum of all interference during the first period, and $v_i'[|v_i'|].y$ contains the sum of all interference during the length of one period for subsequent periods.

3.3.7 Space and Time Complexity

The number of points to calculate (p_i) is quadratic with respect to the number of tasks in the transaction Γ_i (2$|\Gamma_i|$ points for each of the $|\Gamma_i|$ candidate tasks). Thus, storing v_i and v_i' results in a quadratic space complexity since, theoretically, no points from the p_{ic} sets will be removed when calculating p_i.

The method presented in this paper divides the calculation of W_i^* into a pre-calculation and a fix-point iteration phase. A naive implementation of the removal procedure in equation 3.12 requires comparison of each pair of points; resulting in cubic time-complexity ($O(|\Gamma_i|^3)$) for pre-calculating v_i and v_i'.[5] During the fix-point iteration phase, a binary search through a quadratically sized array is performed (either v_i or v_i' in equation 3.12), resulting in $O(\log |\Gamma_i|^2)$ time complexity for calculating W_i^* according to equation 3.7. The original complexity for calculating W_i^* (equation 3.4) is $O(|\Gamma_i|^2)$.

In a complete comparison of complexity, the calculation of $W_i^*(\tau_{ua}, t)$ must be placed in its proper context (see the response time formulas in appendix 3.5). Assume X denotes number of fix-point iterations needed, then the overall complexity for the original approach (equation 3.4) is ($O(X|\Gamma_i|^2)$), whereas our method (equations 3.7 and 3.12) yields ($O(|\Gamma_i|^3 + X \log |\Gamma_i|^2)$). Typically the size of a transaction ($|\Gamma_i|$) is small (less than 100) and the number

[5]In section 3.4 we use an $O(|\Gamma_i|^2 log|\Gamma_i|)$ implementation based on sorting the points and making a single pass through the sorted set.

of fix-point iterations (X) is large (tens or hundreds of thousands), hence our method results in a significant reduction in complexity.

3.4 Evaluation

In order to evaluate and quantify the efficiency (with respect to execution time of RTA) of our proposed method, we have implemented a set of approximate response-time techniques, using the complete set of response-times equations in appendix 3.5. We use these implementations to perform an extensive simulation study. We compare five RTA methods:

- *fast-tight*, presented in this paper and is the method that is optimized the farthest with respect to both analysis speed and tightness. The goal of this simulation study is to quantify its efficiency with respect to execution time of the analysis.
- *fast-slanted*, presented in this paper but without removing the slants (see section 3.3.5). The reason for including it in the analysis is to investigate the impact of reverting back to a stepped stair interference function during response time calculations.
- *tight*, presented in section 3.2 and [MTN04b]. It is only optimized towards tightness. These three methods all produce the exact same tight response times.
- *orig*, presented by Palencia Gutierrez *et al.* [PG98], which is not optimized either for tightness nor for analysis speed. It is included in the evaluation to see if the relative performance degradation of *tight*, compared to *orig*, remains in *fast-tight* when compared to *fast-orig*.
- *fast-orig*, our speed-up method of *orig* presented in [MTN04a]. It is the fastest known RTA for tasks with offsets. It yields the same response times as *orig*. It is included to see if the performance gain of *fast-tight* is comparable to those of *fast-orig*

3.4.1 Description of Simulation Setup

In our simulator, we generate task sets that are used as input to the different RTA implementations. The generated task-sets have the following characteristics:

- Total system load is 90%.
- The number of transactions is 10.
- Jitter (J_{ij}) for each task is 20% of its transaction period.
- Blocking (B_{ij}) is zero.
- The number of tasks/transaction is a variable parameter.
- The priorities are assigned in rate monotonic order.
- Transaction periods (T_i) are randomly distributed in the range 1,000 to 1,000,000 time units (uniform distr.).
- Each offset (O_{ij}) is randomly distributed within the transaction period (uniform distribution).
- The execution times (C_{ij}) are chosen as a fraction of the time between two consecutive offsets in the transaction. The fraction is the same throughout one transaction. The fraction is selected so that the transaction load of 9%.

The execution time for performing the RTA in section 3.4.2 have been obtained by taking the mean value from 50 generated task-sets for each point in each graph. We have measured the execution time on a Pentium 4 laptop. The execution times are plotted with 95% confidence interval for the mean values. Note that, for *fast-orig*, *fast-slanted*, and *fast-tight* the execution times also include the time to perform the pre-calculations presented in Sects. 3.3.4 and 3.3.5.

3.4.2 Simulation Results

Figure 3.9(a) shows how the execution time of the five (although the 3 fast methods are indistinguishable) RTA analysis varies with varying tasks/transaction (all methods are listed in decreasing execution time order). When the number of tasks/transaction is 20, *tight* takes about 86 seconds whereas *fast-tight* takes around 0.63 seconds, which is a speed up of well over two orders of magnitude. Note also that, *tight* has a slight penalty to pay, compared to *orig*, due to more accurate interference modeling.

Zooming in on the three fast analysis methods in figure 3.9(b), we see that *fast-tight* and *fast-orig* are quite comparable in execution times. There are two, mutually opposing, factors that affect their relative timing: The *fast-tight* method shortens its execution time since it sometimes calculates lower response-times than the *fast-orig* method (and hence terminate in fewer fix-point iterations). On the other hand the *fast-tight* method has to spend more time performing pre-calculations and also perform lookup in two different ar-

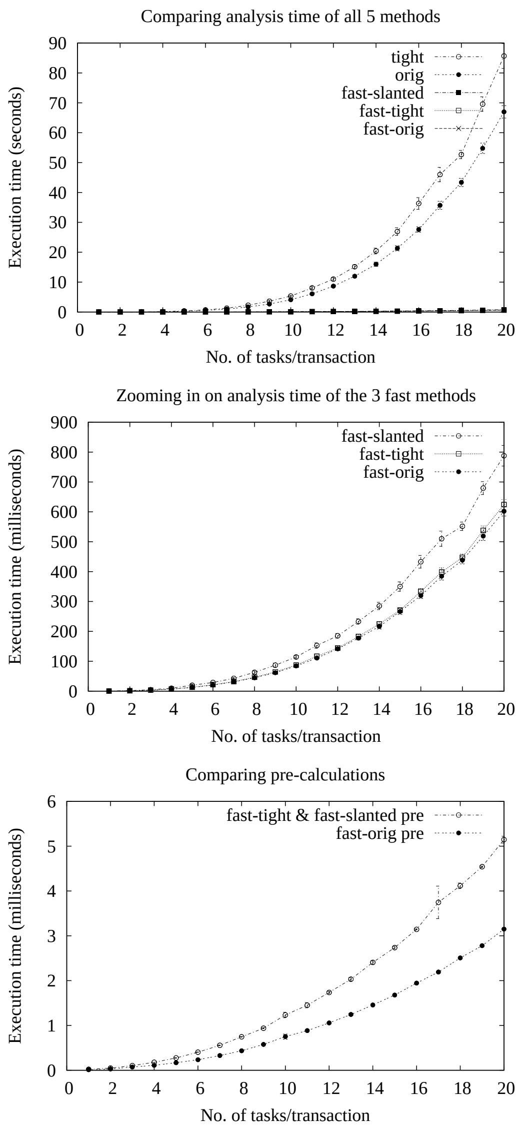


Figure 3.9: Simulation results

rays during each fix-point iteration. In figure 3.9(b) we see that *fast-tight* has consistently slightly longer execution time.

In figure 3.9(b) we also see that *fast-slanted* pays a price of slower fix-point convergence due to the slanted interference function as did *tight* over *orig*. We conclude from figure 3.9(a) and 3.9(b) that the main contribution of speeding up the response times comes from static representation and lookup, but that reverting back to a stepped stair function gives an additional speedup of over 20%.

In figure 3.9(c) we compare the pre-calculations of the three fast methods. Here we can see that the pre-calculations of *fast-tight* and *fast-slanted* is approximately twice that of *fast-orig*. This is expected since they calculate two sets of arrays as opposed to a single set in *fast-orig*. Comparing with figure 3.9(b) one can see that the pre-calculations constitute less than 1% of the total analysis time. One can also discern the complexity of the pre-calculations, and the slope is less steep than what would be expected of a naive implementation with worst-case complexity of $O(|\Gamma_i|^3)$, this is partly due to our (sorting based) $O(|\Gamma_i|^2 log|\Gamma_i|)$ implementation of the pre-calculations, and partly because the worst (theoretical) case, with $|\Gamma_i|^2$ elements in the pre-calculated arrays, never occurs.

We have also simulated an admission control situation. In an admission control situation, a single (low priority) task is added to an (otherwise schedulable) set of already admitted tasks, and its response-time is calculated and compared with its deadline (to decide if the task can be admitted to the system or not). In the admission control the pre-calculation of the already admitted tasks is not included in the execution time. In these simulations, for 20 tasks/transaction, the *tight* method takes about 92 milliseconds whereas the *fast-tight* takes 0.19 milliseconds, which is a speedup with a factor of almost 500. When performing admission control, the speed up in our method is isolated due to two factors: (1) pre-calculations are already done, and (2) no interference from other tasks in the same transaction needs to be accounted for. As can be seen in appendix 3.5, the exact interference-function is used to account for interference from tasks in the same transaction. Since *fast-tight* only improves the approximate interference-function, we isolate our improvement by not needing to account for interference from tasks in the same transaction.

This evaluation shows that combining fast and tight methods for response time analysis, one gets the best of two worlds; a response time analysis method that is both fast and tight, outperforming previous methods by several orders of magnitude.

3.5 Conclusions

In this paper we have presented a novel method that calculates approximate worst-case response times for tasks with offsets. Distinguishing feature of the method is that it calculates tight response times in a short analysis time. We have successfully extended our framework of fast RTA [MTN04a] to be able to apply it to our tight method [MTN04b]. Our improvements are orthogonal and complementary to other proposed extensions to the original offset analysis such as [PG99, Red03].

The main effort in performing RTA for tasks with offsets is to calculate how higher priority tasks interfere with a task under analysis. The essence to calculate fast response times is to find a repetitive pattern and store that pattern statically, and during response time calculations (fix-point iteration), use a simple table lookup. Our tight analysis [MTN04b] exploits the fact that the interference imposed by higher priority tasks is overestimated in traditional RTA. By removing this overestimation, significantly tighter response-times can be calculated. The fast-and-tight analysis presented in this paper successfully does both, resulting in a fast and tight RTA.

Faster RTA has several positive practical implications: (1) Engineering tools (such as those for task allocation and priority assignment) can feasibly rely on RTA and use the task model with offsets, and (2) on-line scheduling algorithms, e.g., those performing admission control, can use accurate on-line schedulability tests based on RTA. Tighter RTA has the practical implications to allow more efficient hardware utilization. Either more functions can be fitted into the same amount of hardware, or less powerful (cheaper) hardware can be used for the existing functions. Hence, our fast-and-tight analysis is a very attractive choice to include in engineering tools and/or admission control software for resource constrained embedded real-time systems.

In a simulation study we see that our novel analysis has very similar computational requirements to that of the fast analysis. Especially we notice that the computational disadvantage of the tight analysis (compared to the original analysis) is completely removed when comparing the fast-and-tight with the fast analysis. Example benchmarks include a speedup of over 100 times for response-time analysis of entire task-sets and a speedup of almost 500 times for single tasks, e.g., corresponding to an admission control situation.

References

[ABD$^+$95] N.C. Audsley, A. Burns, R.I. Davis, K. Tindell, and A.J. Wellings. Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.

[JP86] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[LL73] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[MTN04a] Jukka Mäki-Turja and Mikael Nolin. Faster Response Time Analysis of Tasks With Offsets. In *Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS)*, May 2004.

[MTN04b] Jukka Mäki-Turja and Mikael Nolin. Tighter Response-Times for Tasks with Offsets. In *Proc. of the 10th International conference on Real-Time Computing Systems and Applications (RTCSA'04)*, August 2004.

[PG98] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[PG99] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 328–339, December 1999.

[Red03] O. Redell. Accounting for Precedence Constraints in the Analysis of Tree-Shaped Transactions in Distributed Real-Time Systems. Technical Report TRITA-MMK 2003:4, Dept. of Machine Design, KTH, 2003.

[SH98] M. Sjödin and H. Hansson. Improved Response-Time Calculations. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998. URL: http://www.docs.uu.se/~mic/papers.html.

[Tin92] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

Appendix A: Complete RTA formulae

In this appendix we complete the set of formulas to calculate the worst case response time, R_{ua}, for a task under analysis, τ_{ua}, as presented in Palencia Gutierrez *et al.* [PG98].

The length of a busy period, for τ_{ua}, assuming τ_{uc} is the candidate critical instant, is defined as (Note that the approximation function is not used for Γ_u):

$$\begin{aligned} L_{uac} = & B_{ua} + (p_{L,uac} - p_{0,uac} + 1)C_{ua} + \\ & W_{uc}(\tau_{ua}, L_{uac}) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, L_{uac}) \end{aligned} \qquad \text{(30 in [PG98])}$$

where $p_{0,uac}$ denotes the first, and $p_{L,uac}$ the last, task instance, of τ_{ua}, activated within the busy period. They are defined as:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1 \qquad \text{(29 in [PG98])}$$

and

$$p_{L,uac} = \left\lceil \frac{L_{uac} - \Phi_{uac}}{T_u} \right\rceil \qquad \text{(31 in [PG98])}$$

In order to get the worst case response time for τ_{ua}, we need to check the response time for every instance, $p \in p_{0,uac} \dots p_{L,uac}$, in the busy period. Completion time of the p'th instance is given by:

$$\begin{aligned} w_{uac}(p) = & B_{ua} + (p - p_{0,uac} + 1)C_{ua} \\ & + W_{uc}(\tau_{ua}, w_{uac}(p)) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, w_{uac}(p)) \end{aligned} \qquad \text{(28 in [PG98])}$$

The corresponding response time (for instance p) is then:

$$R_{uac}(p) = w_{uac}(p) - \Phi_{uac} - (p-1)T_u + O_{ua} \qquad \text{(32 in [PG98])}$$

To obtain the worst case response time, R_{ua}, for τ_{ua}, we need to consider every candidate critical instant ,τ_{uc} (including τ_{ua} itself), and for each such candidate every possible instance, p, of τ_{ua}:

$$R_{ua} = \max_{\forall c \in hp_u(\tau_{ua}) \cup a} [\max_{p = p_{0,uac}, \dots, p_{L,uac}} (R_{uac}(p))] \qquad \text{(33 in [PG98])}$$

Appendix B: Proof of Theorem 3.2

We will perform the proof by algebraic manipulation and use braces to highlight the expression that is manipulated in each step. We also annotate braces with the equations, properties, lemmas, or assumptions referred to when performing some manipulations.

When performing the manipulations we will, e.g., rely on the following properties:

(max) — The $\max_v$ operator allows terms that are constant with respect to the maximization variable (v) to be moved outside the maximization operation:

$$\max_v(X_v + Y) = \max_v(X_v) + Y.$$

(sum) — Summation over a set of terms can be divided into two separate summations:

$$\sum_v (X_v + Y_v) = \sum_v X_v + \sum_v Y_v$$

(ceil) — When taking the ceiling ($\lceil\ \rceil$) of a set of terms, terms that are known to be integers can be moved outside of the ceiling expression:

$$X \in \mathbb{N} \Rightarrow \lceil X + Y \rceil = X + \lceil Y \rceil$$

In proving Theorem 3.2 we will use some lemmas.

LEMMA 3.1 *Assume spill tasks are accounted for, then regardless of candidate critical instant c: $I^{Set2}_{ijc}(T_i) = C_{ij}$*

PROOF OF LEMMA 3.1 *For a given critical instant c, perform split according to equation 3.6.*

For an unsplit task τ_{ij} then $\Phi_{ijc} + C_{ij} \leq T_i$ (equation 3.5). For each task $\tau_{ij'}$ and $\tau_{ij''}$ that is the result of splitting then $\Phi_{ij'c} + C_{ij'} \leq T_i$ and $\Phi_{ij''c} + C_{ij''} \leq T_i$ (equation 3.6)

For any task τ_{ij} where $\Phi_{ijc} + C_{ij} \leq T_i$ then

$$\underbrace{I^{Set2}_{ijc}(T_i)}_{\text{Eq.3.2}} = \underbrace{\left\lceil \frac{T_i - \Phi_{ijc}}{T_i} \right\rceil}_{0 \leq \Phi_{ijc} < T_i \text{ (Eq.3.1)}} C_{ij} - \underbrace{x}_{T_i - \Phi_{ijc} \mod T_i \geq C_{ij}}$$

$$= \quad 1C_{ij} - 0 \quad = \quad C_{ij}$$

Hence, for an unsplit task τ_{ij} then $I^{Set2}_{ijc}(T_i) = C_{ij}$. For each task τ_{ij} that is split to $\tau_{ij'}$ and $\tau_{ij''}$, then $I^{Set2}_{ijc}(T_i) = C_{ij'} + C_{ij''} = C_{ij}$

Since this holds for each critical instant c, the lemma holds. □

LEMMA 3.2 *Assume spill tasks are accounted for, and $t = k * T_i + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_i$), then $I^{Set2}_{ijc}(t) = k * I^{Set2}_{ijc}(T_i) + I^{Set2}_{ijc}(t')$*

PROOF OF LEMMA 3.2

$$\underbrace{I^{Set2}_{ijc}(t)}_{\textit{Eq.3.2}} = \underbrace{\left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil}_{\textit{Assumption}} C_{ij} - x =$$

$$\underbrace{\left\lceil \frac{k * T_i + t' - \Phi_{ijc}}{T_i} \right\rceil}\ C_{ij} - x =$$

$$\underbrace{\left\lceil \frac{k * T_i}{T_i} + \frac{t' - \Phi_{ijc}}{T_i} \right\rceil}_{\textit{(ceil)} \wedge k \in \mathbb{N}} C_{ij} - x =$$

$$\underbrace{\left(k + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil \right) C_{ij}} - x =$$

$$k \underbrace{C_{ij}}_{\textit{Lem.3.1}} + \underbrace{\left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij} - x}_{\textit{Eq.3.2}} =$$

$$k * I^{Set2}_{ijc}(T_i) + I^{Set2}_{ijc}(t') \quad \square$$

LEMMA 3.3 *Assume spill tasks are accounted for from $t = 0$, then*

$$T^{ind}_i(\tau_{ua}, T_i) = \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}$$

PROOF OF LEMMA 3.3

$$\underbrace{T_i^{ind}(\tau_{ua}, T_i)}_{Eq.3.9} = \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^{+}(\tau_{ua}, T_i)}_{Eq.3.10} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \Big(\underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + I_{ijc}^{Set2}(T_i) \big)}_{(sum)} - J_i^{ind}(\tau_{ua}) \Big) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \Big(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} + \sum_{\forall j \in hp_i(\tau_{ua})} \underbrace{I_{ijc}^{Set2}(T_i)}_{Lem.3.1} - J_i^{ind}(\tau_{ua}) \Big) =$$

$$\underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \Big(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} + \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} - J_i^{ind}(\tau_{ua}) \Big)}_{(max)} =$$

$$\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1}}_{Eq.3.8} - J_i^{ind}(\tau_{ua}) =$$

$$\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{J_i^{ind}(\tau_{ua}) - J_i^{ind}(\tau_{ua})}_{} = \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} \quad \square$$

THEOREM 3.2 *Assume spill tasks are accounted for, and $t = k * T_i + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_i$), then*

$$T_i^{ind}(\tau_{ua}, t) = k * T_i^{ind}(\tau_{ua}, T_i) + T_i^{ind}(\tau_{ua}, t')$$

PROOF OF THEOREM 3.2

$$\underbrace{T_i^{ind}(\tau_{ua},t)}_{Eq.3.9} = \underbrace{W_{ic}^{+}(\tau_{ua},t)}_{Eq.3.10} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + \underbrace{I_{ijc}^{Set2}(t)}_{Lem.3.2}\big) - J_i^{ind}(\tau_{ua}) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + k * \underbrace{I_{ijc}^{Set2}(T_i)}_{Lem.3.1} + I_{ijc}^{Set2}(t')\big) - J_i^{ind}(\tau_{ua}) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + kC_{ij} + I_{ijc}^{Set2}(t')\big) - J_i^{ind}(\tau_{ua})}_{(sum)} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\Big(\sum_{\forall j \in hp_i(\tau_{ua})} kC_{ij} + \sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')\big) - J_i^{ind}(\tau_{ua})\Big)}_{(max)} =$$

$$\underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} kC_{ij}} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')\big) - J_i^{ind}(\tau_{ua}) =$$

$$k * \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}}_{Lem.3.3} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')\big) - J_i^{ind}(\tau_{ua}) =$$

$$k * T_i^{ind}(\tau_{ua},T_i) + \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} \big(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')\big) - J_i^{ind}(\tau_{ua})}_{Eq.3.10} =$$

$$k * T_i^{ind}(\tau_{ua},T_i) + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua},t')}_{Eq.3.9} =$$

$$k * T_i^{ind}(\tau_{ua},T_i) + T_i^{ind}(\tau_{ua},t') \qquad \square$$

Appendix C: Proof of Theorem 3.3

In proving Theorem 3.3, we will use eq. 28 in [PG98] (see Appendix 3.5), definition of $w_{uac}(p)$, the worst case response time of τ_{ua} with τ_{uc} as the one coinciding with the critical instant, simplified and rewritten as a function of time, $f(t)$:

$$f(t) = K_1 + W_{uc}(\tau_{ua}, t) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, t) \tag{28'}$$

where K_1 is some constant value. We note that a solution to eq. 28' exists, and fix-point convergence is reached, when $f(t) = t$, for some t. Since both exact (W_{uc}) and approximate (W_i^*) interference functions are monotonically increasing, we conclude that $f(t)$ is also monotonically increasing.

LEMMA 3.4 *The smallest solution to eq. 28', denoted s, cannot exist where $f(t)$ has a derivative greater than or equal to 1 (i.e. where $f'(t) \geq 1$).*

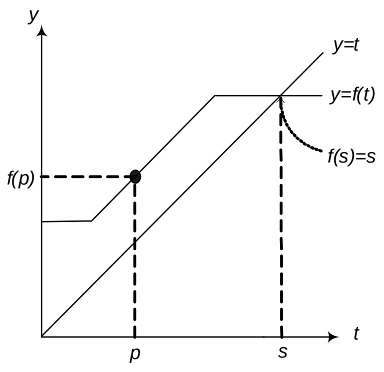


Figure 3.10: Fix-Point Iteration when $f'(t) \geq 1$

PROOF OF LEMMA 3.4 *From [SH98] we know that:*

1. *For any monotonically increasing response-time equation, for any $p < s$, $f(p) > p$ holds.*
2. *We can start fix-point iteration from any point $p < s$ and still find the smallest fix-point s.*
3. *At a point $p < s$ where $f'(p) \geq 1$, consider figure 3.10, the line $y = f(p)$ cannot be converging with line $y = p$ (which has a derivative of 1).*

Assume that s is a point where $f'(t) \geq 1$ then (by the continuousness of $f(t)$) there exists a point $p = s - \epsilon$ (for some small ϵ) where $f'(p) \geq 1$. Then by 1 $f(p) > p$, and by 3 the lines will not be converging. However, by 2 it should be possible to start fix-point iteration at p and converge into s.

A contradiction has been reached and the assumption does not hold. Hence the lemma holds. □

THEOREM 3.3 *Equation 28 in [PG98] cannot have a solution at a time t where any approximate interference function has a derivative greater than or equal to one.*

PROOF OF THEOREM 3.3 *None of the terms in $f(t)$ has a negative derivative. Hence, if for a time t any of the approximate interference functions $W_i^*(\tau_{ua}, t)$ has a derivative of one*[6]*, then the function $f(t)$ has a derivative greater than or equal to one. Then, by lemma 3.4, the theorem holds.* □

[6]The derivative of an approximation function $W_i^*(\tau_{ua}, t)$ is either one (for a slant) or zero (for a stair).

CHAPTER 4

Paper D: Efficient Development of Real-Time Systems Using Hybrid Scheduling

JUKKA MÄKI-TURJA, KAJ HÄNNINEN, AND MIKAEL NOLIN

To appear in the proceedings of the international conference on Embedded Systems and Applications (ESA).

LAS VEGAS USA, JUNE 2005.

Abstract

This paper will show how advanced embedded real-time systems, with functionality ranging from time-triggered control functionality to event-triggered user interaction, can be made more efficient. Efficient with respect to development effort as well as run-time resource utilization. This is achieved by using a hybrid, static and dynamic, scheduling strategy. The approach is applicable even for hard real-time systems since tight response time guarantees can be given by the response time analysis method for tasks with offsets.

An industrial case study will demonstrate how this approach enables more efficient use of computational resources, resulting in a cheaper or more competitive product since more functionality can be fitted into legacy, resource constrained, hardware.

4.1 Introduction

As the complexity of embedded real-time systems keeps growing, both by increases in size and in diversity, the developers are faced with the increasing challenge of modeling, analyzing, implementing and testing both the functional as well as the temporal behavior of these systems. This paper will present ways to simplify some of that complexity by introducing methods to verify the temporal correctness for a larger class of such systems.

Traditionally, one of the design parameters has been what execution model to choose. Two common and widespread execution models are the static and dynamic execution models:

- **Static scheduling**, where a schedule is produced off-line. The schedule contains all scheduling decisions, such as when to execute each task or to send each message. During run-time a simple dispatcher dispatches tasks according to the schedule. Static scheduling is sometimes referred to as time-triggered scheduling.

- **Dynamic scheduling**, where scheduling decisions are made on-line by a run-time scheduler. Typically some task attribute (such as priority or deadline) is used by the scheduler to decide what task to execute. The scheduler implements some queueing discipline, such as fixed priority scheduling or earliest deadline first. Dynamic scheduling is sometimes referred to as event-triggered scheduling.

Since both models have their pros and cons, the design decision of which one to use is not simple. A few trade-offs when choosing execution model are:

- **Overhead** – Since all scheduling and synchronization decision are made off-line in the static approach, the run-time overhead for scheduling is kept low. In dynamic scheduling these decisions are made on-line, often resulting in a larger overhead.

- **Responsiveness** – Statically scheduled systems are inflexible and have therefore limited possibility in responding to dynamic events, resulting in poor responsiveness. Dynamically scheduled systems, on the other hand, handles dynamic events naturally and can provide high degree of responsiveness.

- **Resource usage –** In order to provide some degree of responsiveness for dynamic events in the environment, statically scheduled systems tend to waste resources on redundant polling, whereas event-triggered dynamic schedulers only handle the actual events, enabling better service to soft or non-real time functionality when events do not occur at their maximum rate.

- **Overload** – In static scheduling the effects of overload are highly predictable. The exact capacity, e.g. in terms of number of inputs handled, is known and the effect of lost events, e.g. due to slow polling, can be predicted. In dynamic scheduling, no natural overload control is inherent. Instead, ad-hoc mechanisms are used to prevent, e.g., faulty sensors from flooding the systems with interrupts. A dynamically scheduled system which becomes overloaded is unpredictable, it is often difficult to assess which buffer will overflow and thus which tasks will miss their deadlines.

- **Determinism** – A statically scheduled system is highly deterministic, it executes according to the pre-defined schedule each time. A dynamically scheduled system, on the other hand, may exhibit different behavior each time the system is run, due to, e.g., race conditions on shared resources. This has a major impact on reproducibility, and thus also on the functional testability, of the system. Determinism also simplifies the verification process which is a major part when certifying safety critical applications.

- **Complex constraints** – Statically scheduled systems can handle more complicated inter-task relation constraints. For example, control systems, where control performance is important, need to have small (input and/or output) jitter, which is easier to accommodate in a static scheduler than with simpler dynamic scheduling parameters.

- **Adding new functionality** – Once a static schedule has been constructed it can be very hard to add new functionality in the system, a completely new schedule has to be constructed. For a dynamically scheduled system, new functions can be added with a minimum of impact on other parts of the system.

For further discussions on these trade-offs see [Loc92] which advocates cyclic scheduling), and [XP00] which advocates dynamic, fixed priority, scheduling.

As can be seen, both approaches have their virtues and one often wishes to have both approaches available when developing embedded real-time applications. This desire is clearly illustrated by the last few years of development in the area of field busses for automotive applications. The Controller Area Network (CAN) [CAN92] has been predominant in the automotive industry. CAN provides dynamic scheduling (using fixed priority scheduling). However, the automotive industry felt a need for a more dependable and predictable bus architecture. So when Kopetz brought attention to his Time Triggered Protocol (TTP) [KG94], which provides static scheduling, many automotive manufacturers and their sub-contractors embraced the new technology. It was soon recognized that TTP was a bit *too* static. Hence, a consortium of automotive manufacturers and sub-contractors started the development of FlexRay [Flx], which provides both static and dynamic scheduling. Also, on the operating-system side, products that support both static and dynamic scheduling have emerged. For instance, Arcticus Systems' operating system Rubus [Arc], and the open source real-time operating system Asterix [Ast]. In fact, most priority driven operating systems can implement hybrid static and dynamic scheduling by letting a dispatcher (a time-table) execute at highest priority.

Thus, we see that the need to combine static and dynamic scheduling have led to some practical solutions available today. However, one problem with systems that tries to combine static and dynamic scheduling is that they often consider the dynamic part as non real-time, e.g. [Arc, Flx]. That is, dynamic scheduled tasks/messages are not given any response-time guarantees, only best-effort service is provided. However, in order to fully utilize the potential of combining static and dynamic scheduling in hard real-time systems, both the dynamic and the static parts need to be able to provide response-time guarantees. A recent study of industrial needs recognizes that one of the key issues for embedded systems is analyzability [MFN04].

This paper presents a method to model hybrid, statically and dynamically, scheduled systems with the task model with offsets [MTN04]. With this model, and the corresponding response time analysis, tight response time guarantees can be given also for dynamically scheduled tasks. The modelled system can be realized with commercially available operating systems support. Furthermore, in a case study we show how a legacy system at Volvo Construction Equipment could benefit from this approach by migrating functionality from the resource demanding statically scheduled part to the dynamically scheduled part, freeing system resources while still fulfilling original temporal constraints.

Paper Outline: Next, section 4.2 describes the type of systems studied in this paper. Section 4.3 shows how these systems can be modelled using the task model with offsets. Section 4.4 discusses related work. Section 4.5 illustrates, through a case study, how this approach can be applied to a legacy system, migrating functions from a static schedule, freeing system resources. Finally, section 4.6 presents our conclusions.

4.2 System description

In this paper, we address the issue of providing tight response-time guarantees to dynamically scheduled tasks running “in the background” of a static schedule. The system model contains:

- **Interrupts**. There may be multiple interrupt levels, i.e., an interrupt may be preempted by higher level interrupts.

- **A static cyclic schedule**.

  - A set of periodic static tasks (functions) are scheduled in the schedule. Each task has a known worst case execution time (WCET).
  - The schedule has a length (a duration) that is equal to the LCM (least common multiple) of all statically scheduled function periods. The schedule is constructed off-line by a scheduling tool.
  - Each function is scheduled at an offset relative to the start of the schedule. This is also referred to as a function’s *release time*.
  - The static cyclic scheduler activates each function in the schedule at its release time. When the whole schedule has been executed the schedule is restarted from the beginning.

  Interrupts may preempt the execution of statically scheduled functions.

- **A set dynamically dispatched tasks**. We call each such task a *dynamic task*. These tasks executes in the time slots available between interrupts and statically scheduled functions. Dynamic tasks are scheduled by a fixed priority preemptive scheduler. They are assumed to be periodic or, at least, to have a known minimum time between two invocations.

We assume that a static cyclic schedule has been constructed prior to the analysis of dynamic tasks. Furthermore, we assume that the schedule is valid even if its functions are preempted by interrupts. How a scheduler can generate a feasible schedule, with interfering interrupts, is described in [SEF98].

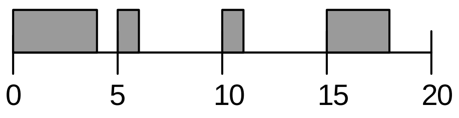


Figure 4.1: Example of static cyclic schedule

4.2.1 Example system

Figure 4.1 shows a static cyclic schedule of length 20, with 4 functions released at times 0, 5, 10 and 15, with WCETs 4, 1, 1 and 3 respectively.

In figure 4.2 we see an example execution scenario when executing the schedule from figure 4.1, with one interfering interrupt source and one dynamically scheduled task (two instances of that task are activated). We make the observation that both interrupts and the static schedule act like higher priority tasks from the dynamic tasks' point of view.

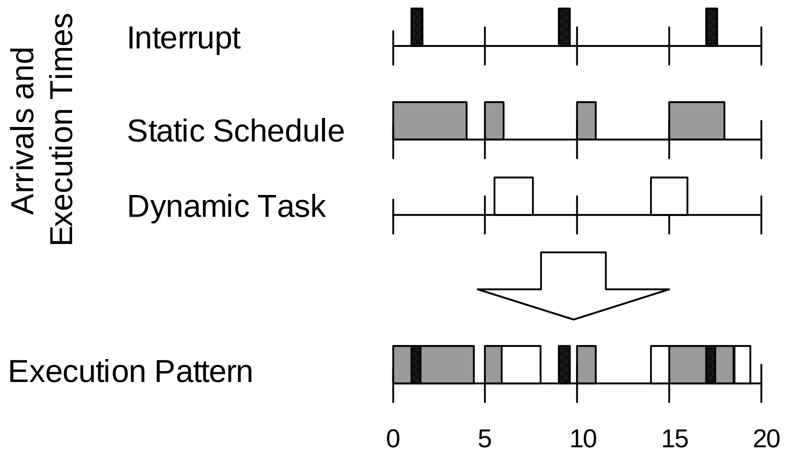


Figure 4.2: Example execution scenario

One of the main objectives of this paper is to enable response-time calculations for dynamic tasks. The goal is to model static schedules (and interrupts) so as to incur as little interference on dynamic tasks execution as possible. Thus, modeling both functions' WCETs as well as their release times as accurately as possible.

4.3 Modeling the system

Classical response-time analysis (see e.g. [ABD$^+$95, BW96, JP86]), assumes that a critical instant[1] occurs when all tasks are released simultaneously. Using this model, the static schedule described in section 4.2, can be modelled as 4 tasks. These tasks would have a period of 20 and WCETs of 4, 1, 1, and 3 respectively. However, this approach is overly pessimistic since it assumes that all four static tasks can be released for execution at the same time. In our example, assuming no interrupt interference, a dynamic task with a WCET of 1, would have a response time of 10 (4+1+1+3+1). However, looking at figure 4.1 one can see that the actual worst possible response-time is 5 (if the dynamic tasks coincides with the static function scheduled at time 0).

In static schedules it is impossible for all static tasks to start at the same time. The task model with offset introduced by [PG98, Tin92] is able to capture the time separation in static schedules, and thus reduce the pessimism. In [MTN04] we further reduced the pessimism in the corresponding response time formulae.

4.3.1 Task model with offsets

The task set, Γ, in [MTN04] consists of a set of k transactions, $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i. A transaction Γ_i, contains $|\Gamma_i|$ number of tasks, and each task is activated when a relative time, *offset*, elapses after the arrival of the event.

τ_{ij} is used to denote a task. The first subscript denotes which transaction the task belongs to, and the second subscript denotes the number of the task within that transaction. A task τ_{ij} is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The task set Γ is formally expressed as follows:

$$\begin{aligned} \Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\} \\ \Gamma_i &:= \langle \{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i \rangle \\ \tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} \rangle \end{aligned}$$

There are no restrictions placed on offset, deadline or jitter. The maximum blocking time for a task, τ_{ij}, is the maximum time it has to wait for a resource

[1]Point in time, where the task under analysis is released for execution, resulting in the longest possible response-time.

which is locked by a lower priority task. In order to calculate the blocking time for a task, usually, a resource locking protocol like priority ceiling or immediate inheritance is needed. Algorithms to calculate blocking times for different resource locking protocols are presented in [But97]. Priorities can be assigned with any method (e.g. rate monotonic, deadline monotonic, or user defined priorities). One must assume that the load of the task set is less than 100%.[2]

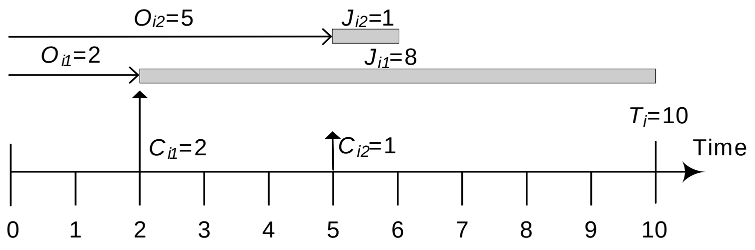


Figure 4.3: Example transaction

Parameters for an example transaction (Γ_i) with two tasks (τ_{i1} and τ_{i2}) is depicted in figure 4.3. The offset denotes the earliest possible release time of a task relative to the start of its transaction and jitter (illustrated by the shaded region) denotes maximum possible variability in the actual release of a task. The upward arrows denotes earliest possible release of a task and the height of the arrow corresponds to the amount of execution released. The end of the shaded region represents the latest possible release of a task.

4.3.2 System model

The system in section 4.2 can be modelled, and dynamic tasks subsequently analyzed for response times, with the above task model as follows (subscripts i, s, and d denote a generic interrupt, static, and dynamic transaction respectively):

- **Each interrupt** will be modelled as a transaction, Γ_i, containing one single task (i.e., $|\Gamma_i| = 1$) with T_i set to minimum inter-arrival time of the corresponding interrupt. These interrupt tasks will have the highest

[2]This can easily be tested, and if not fulfilled some response-times may be infinite; rendering the task set unschedulable.

priorities in the system. If there are several interrupt levels, priorities are assigned accordingly, i.e., highest priority to highest interrupt level.

- **The static schedule** is modelled as one transaction, Γ_s, where each release time in the schedule is modelled as one task, τ_{sj}, where the offset ,O_{sj}, is set to the corresponding release time. The worst case execution time, C_{sj}, is set to the corresponding functions WCET. The priority, one suffices, for static tasks must be lower than for any interrupt, but higher than those for dynamic tasks.

  Our example schedule of figure 4.1 will be modelled as a transaction ($T_s = 20$) with 4 tasks, with offsets 0, 5, 10, 15 and worst case execution time of 4, 1, 1, 3 respectively.

- **Dynamic tasks** will have the most variability on how they are modelled. In the simplest case they are modelled exactly the same way as interrupts but with lower priorities. This situation corresponds to simple periodic (or sporadic) dynamic tasks with no jitter, no time separation (offsets), and no blocking. However depending on the nature of the dynamic tasks their corresponding transaction can be extended by:

  – jitter if there is variability in their periodicity,

  – by blocking if they share resources and providing the run-time system supports an analyzable resource sharing protocol, and

  – offsets if there are temporal dependencies, such as precedence, among dynamic tasks.

  Note that dynamic tasks cannot communicate with static tasks, via locked resources, since they must not affect their temporal behavior. However, there exist methods to communicate between these two systems that will not affect the temporal behavior of static tasks, see e.g. [NNT$^+$04].

  Assuming the dynamic task of figure 4.2 is a sporadic task with minimum inter-arrival time of 10 time units and a release jitter of 3 time units, it is modelled as a transaction with $T_d = 10$ containing one task with $J_{dj} = 3$. The execution time is 2 and since it is the lowest priority task the blocking is zero ($C_{dj} = 2$ and $B_{dj} = 0$).

The formulae to calculate the response times rely on a relaxed critical instant assumption stating that only one task out of every transaction has to coincide with the critical instant. The complete formulae can be found in [MTN04], and would, for our example system of figure 4.2, result in a response time of 5 time units for a dynamic task with $C_{dj} = 1$, assuming no interrupt interference.

Since all type of tasks, interrupt, static, and dynamic, can be analyzed for responsiveness, the inability of providing response time guarantees will no longer be a basis for rejecting an execution model for a function, thus making hybrid static and dynamic scheduling suitable even for hard real-time systems.

4.4 Related work

There has been number of research projects addressing the issue of combining several execution models [RS01, SRLK02, BBLB03]. These provide reservation-based guarantees where task characteristics are not fully known in advance. Furthermore, no commercially available real-time operating system support exist for them. Our approach is to model existing systems, supported by commercial RTOSes, where task attributes are fully known at design time. However, [RRW$^+$03] aims at modeling real situations through hierarchically modeling different schedulers. They cover preemptive and non-preemptive priority schedulers and do not model static schedulers. In fact, the work presented in this paper could extend their more general framework with the ability to model also static schedulers.

4.5 Case Study

A case study [HR03] conducted at Volvo Construction Equipment (VCE) [Vol], with the objective of finding a way to use available resources in a more efficient way has studied the design trade-offs between static and dynamic scheduling.

VCE has a tradition in statically scheduled systems. This is mainly due to the safety critical nature of their control systems in their heavy machinery, e.g., articulated haulers, trucks, wheel loaders and excavators. Rubus OS by Arcticus [Arc], used by VCE, has run-time support for the system model described in section 4.2.

Currently at VCE, all safety critical functionality is implemented in the static part and only soft real-time or non real-time activity resides in the dynamic part. In recent interviews (in an ongoing research project) they state that about 20-25% of their applications are considered safety critical, mainly residing in transmission and engine control. However, some operational modes, have static schedule utilization as high as 74%.

The demand on more functionality in next generation machinery is growing. However, the static schedule is getting close to full utilization, leaving little or no room for new functionality. This can either be addressed with new and more expensive hardware or to find a better way of utilizing the current hardware resources.

Demand on responsiveness (i.e. deadlines) for functionality in the static part ranges from a few milliseconds up to several seconds. This could potentially result in very large schedules (with corresponding high memory consumption). VCE's solution to this has been to fix the schedule length at 100ms, which result in waste of computing resources due to redundant polling for any function with a responsiveness demand higher than 100ms (even functions with responsiveness demand within 100ms but associated with events that occur seldom will in this case waste computing resources). A solution that could get rid of this redundant polling, while still guaranteeing the responsiveness and without increasing the schedule length, would be highly desirable.

4.5.1 An example system

Here we will present an example system that can be viewed as a simplified version of one of the systems constructed by VCE. A complete system would consists of several hundreds of tasks [HR03] and would be too complex to present in this paper. We will show how functions currently residing in the static part can be moved to the dynamic part and, by using the response-time analysis of [MTN04], still guarantee that the function deadlines will be met. Type of functionality that could be moved, according to [HR03], consists of events that by nature are event-triggered, visual interaction with driver, and logging of operational statistics. Another example of functionality that may be moved to the dynamic part is control functionality that is not part of sampling or actuation. Control performance is often sensitive to jitter in sampling and actuation and therefore often placed in a static schedule [Cer99]. However, the control calculation and updating of control state do not have these strict requirements on jitter and their responsiveness requirement is only restricted by the corresponding output action and sampling in the next period respectively. Therefore control and updating control state functionality could be moved to the dynamic part.

For our example, the task specification in table 4.1 on the next page will be used. (For simplicity we will in this example ignore interrupt interference.) Tasks F and G handle events that may occur once every 2000ms, and with a response time requirement of 100ms. Placing tasks F and G in a static schedule,

Task i	T_i	C_i	D_i	U^{100}	U^T
A	10	2	10	20%	20%
B	20	2	5	10%	10%
C	50	1	2	2%	2%
D	50	6	50	12%	12%
E	100	8	100	8%	8%
F	2000	7	100	7%	0.35%
G	2000	8	100	8%	0.4%
H	2000	8	2000	8%	0.4%

Table 4.1: The set of tasks in the Static system

means that they would have to be polled at the rate of their deadline (100ms) instead of their period (2000ms) (since we do not know exactly when the events are going to occur). Task H, however, could be polled at the rate of its period (2000ms), however, the resulting schedule would become too large and memory consuming (it would have to extend for 2000ms and thus consume over 20kb of ROM). Setting the schedule length to 100ms would be adequate for all tasks except task H. Hence, the schedule length is set to 100ms, and a resulting schedule can be seen in figure 4.4.

In table 4.1, U^{100} represents the task utilization when scheduled in a static schedule with a period of 100ms, and U^T represents the utilization when tasks are scheduled with their period.

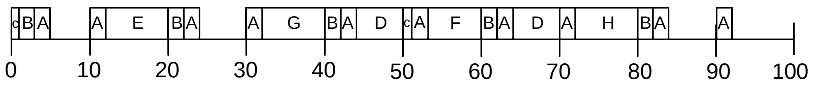


Figure 4.4: Static schedule for table 4.1 task set

The total utilization of the static schedule is 75%. Adding new functionality, requiring some kind of temporal guarantee, to this system can be difficult, there are not many free time-slots in the schedule, especially if there has to be room also for interrupts and non-real-time functionality.

Improving the system

However if tasks F, G, and H could be made event triggered, by placing them in the dynamic part of the Rubus OS, some resources could be freed. The resulting static schedule can be seen in figure 4.5. The utilization for the static schedule now becomes 52%. The utilization for the three dynamic tasks are 1,15%, resulting in a total utilization of just above 53%. Thus, by moving these three tasks from the static schedule we free nearly 22%[3] of the CPU resources.

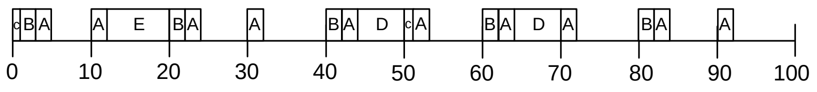


Figure 4.5: Schedule without tasks F, G and H

Now, it remains to see whether the three tasks will meet their deadlines when running as dynamic tasks. To be able to calculate response times for tasks F, G, and H we model the static schedule as a transaction with $T_s = 100$. WCETs and offsets are set as follows:

$$C_{sj} = (5,\ 10,\ \ 4,\ \ \ 2,\ 10,\ \ 3,\ 10,\ \ \ 2,\ \ 4,\ \ 2)$$
$$O_{sj} = (0,\ 10,\ 20,\ 30,\ 40,\ 50,\ 60,\ 70,\ 80,\ 90)$$

Assuming that F, G, and H have priorities high, medium, and low respectively, we can calculate the response times for the three tasks according to [MTN04]. And the result is:

$$R_F = 26 \quad R_G = 44 \quad R_H = 64$$

We see that all three tasks will meet their deadlines of table 4.1. In fact, their responsiveness is considerably increased compared to being statically scheduled every 100ms. It could be mentioned that by removing tasks F, G and H from the schedule we have enabled shorter response times for other dynamic tasks, that might have existed in the system, as well. The schedule in figure 4.4 has a longest busy period of 54ms (between 30–84), whereas the new schedule in figure 4.5 has a longest busy period of 14ms (between 10–24). Since any dynamic task (in the worst case) will have to wait for the longest busy period, we now have significantly reduced that time.

With the approach presented in this paper the static schedule could be kept small (with respect to memory consumption as well as utilization). By modeling the static schedule as one transaction, response time analysis for task with offsets can be used to evaluate timeliness for the dynamic part.

[3]Increase in overhead for tasks F, G, and H as dynamic tasks will be marginal, hence not considered here.

Our solution reduce utilization by moving functionality, previously polled excessively, from the static schedule to the dynamic part. Our method also gives a possibility to shrink the static schedule since functions with long periods can be moved from the static schedule. It should be mentioned however, that all tasks in the static schedule share a common stack, whereas moving tasks from the schedule to the dynamic part may require them to have separate stacks, hence increasing the memory consumption for dynamic tasks. However, using a resource locking protocol such as the immediate inheritance allows also dynamic tasks to share a single stack [But97, Nor99].

The possibility to selectively migrate functions from static scheduled legacy systems to dynamic scheduled systems will substantially facilitate for companies to gradually move into the area of dynamic scheduling, and thus, in the long run, help companies to use cheaper hardware for, or fit more functions into, their products. Also the development process becomes easier because event triggered functionality does not have to be force-fitted into a static model.

4.6 Conclusions

As stated in [MFN04] analyzability is one of the major concern for embedded systems development. We have in this paper shown how a hybrid, static and dynamic, scheduling model can be modelled and dynamic tasks analyzed for responsiveness. The type of system presented can be realized by commercially available OS support, e.g., Rubus OS by Arcticus [Arc]. In fact, any fixed priority OS complemented with an external static scheduler can implement this type of system with the static schedule as a task at highest priority.

A hybrid, static and dynamic, scheduling model simplifies the design tradeoffs of which scheduling model to choose. Appropriate scheduling model can be chosen on function level instead of system level. Since temporal guarantees can be provided, this approach will also be applicable for hard real-time systems. Choosing the most appropriate model for each function, instead of force-fitting it to an overall model, not only simplifies the design choices but also gives the possibility to save system resources and improve responsiveness. This is demonstrated in a case study [HR03] at Volvo Construction Equipment using the commercial real-time operating system Rubus by Arcticus [Arc].

References

[ABD$^+$95] N.C. Audsley, A. Burns, R.I. Davis, K. Tindell, and A.J. Wellings. Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.

[Arc] Arcticus Systems Web-Page. http://www.arcticus.se.

[Ast] The Asterix Real-Time Kernel. http://www.mrtc.mdh.se/projects/-asterix/.

[BBLB03] Scott Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time, and Non-Real-Time Processes. In *Proc. 24th IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, December 2003.

[But97] G.C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, 1997. ISBN 0-7923-9994-3.

[BW96] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, second edition, 1996. ISBN 0-201-40365-X.

[CAN92] Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High Speed Communications, February 1992. ISO/DIS 11898.

[Cer99] A. Cervin. Improved scheduling of control tasks. In *Proc. of the 11th Euromicro Workshop of Real-Time Systems*, pages 4 – 10, June 1999.

[Flx] FlexRay Home Page. http://www.flexray-group.org/.

[HR03] K. Hänninen and T. Riutta. Optimal Design. Master's thesis, Mälardalens Högskola, Dept of Computer Science and Engineering, 2003.

[JP86] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[KG94] H. Kopetz and G. Grünsteidl. TTP – A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[Loc92] C.D. Locke. Software Architecture For Hard Real-Time Applications - Cyclic Executives vs. Fixed Priority Executives. *The Journal of Real-Time Systems*, 4:37–53, 1992.

[MFN04] Anders Möller, Joakim Fröberg, and Mikael Nolin. Industrial Requirements on Component Technologies for Embedded Systems. In *7th International Symposium on Component-based Software Engineering (CBSE7)*. IEEE Computer Society, May 2004.

[MTN04] Jukka Mäki-Turja and Mikael Nolin. Tighter Response-Times for Tasks with Offsets. In *Proc. of the 10th International conference on Real-Time Computing Systems and Applications (RTCSA'04)*, August 2004.

[NNT$^+$04] Dag Nyström, Mikael Nolin, Aleksandra Tesanovic, Christer Norström, and Jörgen Hansson. Pessimistic Concurrency-Control and Versioning to Support Database Pointers in Real-Time Databases. In *Proc. of the 16th Euromicro Conference on Real-Time Systems*, June 2004.

[Nor99] Northern Real-Time Applications. SSX5 True RTOS, 1999.

[PG98] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[RRW$^+$03] J. Regher, A. Reid, K. Webb, M. Parker, and J. Lepreau. Evolving real-time systems using hierarchical scheduling and concurrency analysis. In *Proc. 24th IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, December 2003.

[RS01] J. Regher and J.A. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proc. 22th IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, December 2001.

[SEF98] K. Sandström, C. Eriksson, and G. Fohler. Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System. In *Proc. of the 5th International conference on Real-Time Computing Systems and Applications (RTCSA'98)*, 1998.

[SRLK02] S. Saewong, R. Rajkumar, J.P. Lehoczky, and M.H. Klein. Analysis of hierarchical fixed priority scheduling. In *Proc. of the 14th*

Euromicro Conference on Real-Time Systems. IEEE Computer Society, June 2002.

[Tin92] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

[Vol] Volvo Construction Equipment. http://www.volvoce.com.

[XP00] J. Xu and D.L. Parnas. Priority Scheduling Versus Pre-Run-Time Scheduling. *The Journal of Real-Time Systems*, 18(1):7–23, January 2000.

III

Appendicies

APPENDIX A
RTA Formulae

This appendix presents RTA formulae for some of the extensions made to the basic RTA presented by [JP86]. The task model for basic RTA is as follows. A task τ_i is specified by:

- A period, T_i.
- Worst case execution time, C_i.
- Deadline, D_i.
- Priority, P_i.

The following **assumptions** must hold in order for the analysis to be valid:

- There can be no synchronization between tasks.
- Tasks must not suspend themselves.
- Deadlines must be less or equal to corresponding periods, i.e., $D_i \leq T_i$.
- Tasks must have unique priorities.

The response time for task τ_i, R_i, consists of τ_i-s own worst case execution time and higher priority task interference:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Where $hp(i)$ denotes the set of tasks having higher priority than τ_i.

The extensions presented in this appendix aim at increasing the applicability of RTA either by extending the task model or lifting some of the restrictions in the assumptions.

Adding blocking

Assuming the blocking factor, the time a task has to wait for a shared resource held by a lower priority task, can be bound to B_i, the response time, for τ_i, can then be obtained by:

$$R_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Adding jitter

Assuming there is a uncertainty of the periodic release of a task, jitter, by an amount of J_i, the response time can be obtained by:

$$\begin{aligned} w_i &= C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \\ R_i &= w_i + J_i \end{aligned}$$

Adding offsets – approximate approach

Adding offset relations to the basic model means that all tasks can no longer be released for execution at the same time. Tasks with mutual offset relations are grouped into transactions (Γ_i), each with their own period time (T_i). The definition of critical instant is relaxed to: One task out of every transaction is released at the critical instant. This means that there only can be a simultaneous release of some tasks. Task τ_{ij} denotes task j of transaction Γ_i. Offset O_{ij} denotes the release of a task relative to the start of the transaction.

Assuming one knows what task out of every transaction coincides with the critical instant, let us call such a task a critical instant task τ_{ic}, the phasing between the critical instant task and any other task can be calculated as:

$$\Phi_{ijc} = (O_{ij} - O_{ic}) \mod T_i$$

That is, Φ_{ijc} denotes the time of a subsequent (or perhaps simultaneous) release of τ_{ij}, relative to the release of τ_{ic} at the critical instant.

With τ_{ic} as the critical instant task of Γ_i, the interference Γ_i poses on a task under analysis, τ_{ua}, during time t, is:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij}$$

Where $hp_i(\tau_{ua})$ denotes the set of tasks with higher priority to τ_{ua} and which belongs to transaction Γ_i.

Since one can not know which task coincides with the critical instant beforehand, the exact approach must examine every possible combination of critical instant tasks in the different transactions. This becomes computationally intractable for anything but small task sets. Therefore an approximate interference of Γ_i, considering each transaction in isolation, is given by:

$$W_i^*(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t)$$

With this exact and approximate interference, the response time for a task under analysis (R_{ua}) can calculated as:

$$R_{ua} = C_{ua} + \max_{\forall c \in hp_u(\tau_{ua}) \cup a} \Big(\sum_{\forall i \neq u} W_i^*(\tau_{ua}, R_{ua}) + W_{ic}(\tau_{ua}, R_{ua}) \Big)$$

With this definition we see that the approximate interference function is used for all but τ_{ua}-s own transaction Γ_u. For Γ_u the exact interference is used and thus one must examine each task (including τ_{ua}) in Γ_u as coinciding with the critical instant.

Communications device – CAN

CAN is a non-preemptive fixed priority resource where a message can only be interrupted during the time it takes to send one bit, τ_{bit}, over the communication channel. The response-time formulae thus becomes:

$$w_i = \tau_{bit} + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i}{T_j} \right\rceil C_j$$

$$R_i = w_i + C_i - \tau_{bit}$$

Response time larger than period

Assuming the response time is allowed to be greater than the period for τ_i, implies that there can be several active task instances of τ_i active simultaneously. In calculating the response time, another iterative step is added. First, the length of level-i busy period (processor is busy executing tasks with priority higher or equal to τ_i) is determined. The response time for each task instance, in that busy period, has to be calculated, and the maximum of them constitutes the worst case response time for τ_i:

Length of level-i busy period:

$$BP_i = \left\lceil \frac{BP_i}{T_i} \right\rceil C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{BP_i}{T_j} \right\rceil C_j$$

Obtaining the worst case response time:

$$R_i = \max_{k \in 1 \ldots \left\lceil \frac{BP_i}{T_i} \right\rceil} R_{i,k} \quad \text{where}$$

$$R_{i,k} = kC_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_{i,k}}{T_j} \right\rceil C_j - (k-1)T_i$$

APPENDIX B
Table of concepts

Concept	Meaning
Activation time	The time of the event occurrence triggering a task.
Blocking time	The time a task under analysis must wait for a lower priority task to release a shared resource.
Critical instant (c.i.)	Point in time leading to maximum response time for a task under analysis.
Imposed interference	Interference actually imposed on a task during a time interval.
Jitter	Difference between earliest and latest possible release of a task.
Offset	Time difference between the earliest and latest possible release of a task.
Released for execution interference	Interference of tasks released for execution, i.e., sum of WCET of tasks placed in the ready queue.
Release time	The time the task is released for execution, i.e., placed in the ready queue.
Response time	Time from activation to completion of a task.
Self suspension	A task voluntarily suspending itself, e.g., by a delay call.
Set 1 task instances	Task instances released before c.i. and delayed by jitter to be released at c.i.
Set 2 task instances	Task instances released after the c.i.
Worst case execution time	Longest possible execution time of a task if it could run uninterrptedly on the CPU.

APPENDIX C
Table of abbreviations

Abbreviation	**Meaning**
BCRT	Best case response time
c.i.	Critical instant
EDF	Earliest deadline first
ET	Event-triggered
FPS	Fixed-priority scheduling
LCM	Least Common Multiple
MRTC	Mälardalen Research and Technology Centre
RM	Rate Monotonic
RTA	Response-time analysis
TT	Time-triggered
TTP	Time-Triggered Protocol
VCE	Volvo Construction Equipment
WCRT	Worst case response time
WCET	Worst case execution time

APPENDIX D
Table of symbols

Symbol	**Meaning**
B_{ij}	Blocking factor of task τ_{ij}
C_{ij}	WCET of task τ_{ij}
D_{ij}	Deadline of task τ_{ij}
Γ_i	Transaction i
I_{ijc}^{Set1}	Amount of interference of task instances in Set 1
$I_{ijc}^{\mathit{Set2}}(t)$	Amount of interference of task instances in Set 2 during t
J_{ij}	Jitter of task τ_{ij}
$J_i^{ind}(\tau_{ua})$	Jitter induced part of the approximate interference of Γ_i
O_{ij}	Offset of task τ_{ij}
P_{ij}	Priority of task τ_{ij}
Φ_{ijc}	Relative phasing between critical instant task τ_{ic} and τ_{ij}
R_{ij}	Response time of task τ_{ij}
R_i^k	The kth fix-point iteration in calculating R_i
T_i	Period of task τ_i or transaction Γ_i
T_i^c, T_i^t	Arrays representing $T_i^{ind}(\tau_{ua}, t)$ up to T_i. $T_i^c[n]$ is the maximum amount of interference up to time intervals of length $T_i^t[n]$
τ_{ua}	Generally: Task a of transaction Γ_u. Also, task *under analysis*
τ_{ic}	Critical instant task, i.e., the task in Γ_i that coincides with the c.i.
$T_i^{ind}(\tau_{ua}, t)$	Time induced part of the approximate interference of Γ_i
v_i	Point set representing $T_i^{ind}(\tau_{ua}, t)$ up to T_i. $v[n].y$ is the amount of interference up to time intervals of length $v[n].x$
v_i'	Point set representing $T_i^{ind}(\tau_{ua}, t)$ for $T_i < t \leq 2T_i$
$W_i^*(\tau_{ua}, t)$	Approximate interference imposed on τ_{ua} by Γ_i during t
$W_{ic}(\tau_{ua}, t)$	Exact interference imposed on τ_{ua} by Γ_i during t, assuming τ_{ic} coincides with the critical instant