Real-time scheduling for media processing using conditionally guaranteed budgets

Citation for published version (APA):
Bril, R. J. (2004). *Real-time scheduling for media processing using conditionally guaranteed budgets*. [Phd Thesis 2 (Research NOT TU/e / Graduation TU/e), Frits Philips Inst. Quality Management]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR580445

DOI:
10.6100/IR580445

Document status and date:
Published: 01/01/2004

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Real-time scheduling for media processing using conditionally guaranteed budgets

Printed by: Eindhoven University Press, Eindhoven, The Netherlands.
Cover design: Hennie Alblas (Philips Research AV-service) and Reinder J. Bril.
The drawing on the cover has been made by Joëlla V. Bril, Marijn J. Bril, and Wander J. Bril, and illustrates that budgets cannot be guaranteed without enforcement.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Bril, Reinder Jaap.

Real-time scheduling for media processing using conditionally guaranteed budgets/ by Reinder Jaap Bril. -
Eindhoven: Technische Universiteit Eindhoven, 2004. Thesis -
ISBN: 90-74445-62-4
Subject headings: real-time scheduling, media processing, Quality-of-Service, resource budgets.

Real-time scheduling for media processing using conditionally guaranteed budgets

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. R.A. van Santen, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op woensdag 29 september 2004 om 16.00 uur

door

Reinder J. Bril

geboren te Uithuizen (Gr.)

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. E.H.L. Aarts (Technische Universiteit Eindhoven, The Netherlands)
en
prof.dr. G. Fohler (Mälardalen University, Sweden)

de copromotor:

dr.ir. W.F.J. Verhaegh (Philips Research Eindhoven, The Netherlands)

en de overige leden van de kern commissie:

prof.dr.-ing.habil. C. Hentschel (University of Technology Cottbus, Germany)
en
dr. J.J. Lukkien (Technische Universiteit Eindhoven, The Netherlands).

The work described in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).
IPA Dissertation Series 2004-13

To my beloved wife Carmen,
daughters Joëlla and Marijn,
and son Wander.

Preface and Acknowledgement

"A fanatic is one who can't change his mind and won't change the subject."

Winston Churchill (1874 - 1965)

This thesis is an important milestone for me towards a scientific career, and enables a long cherished dream of mine to come true. Ever since September 1983, even before my M.Sc. graduation, I have aspired to work in an environment that either provides me ample opportunities to perform research myself or allows me to maintain close contacts with researchers. During the past twentyone years, I was blessed by being able to realize that aspiration; I have worked at universities and industrial research organizations for almost thirteen years, and in (research and) development organizations for eight years. Since the start of my professional career in January 1984, I have considered topics at the edge of the unexplored as attractive and challenging. One way or another, such topics always seem to present themselves, or, alternatively, I may just be sensitive to be attracted by such topics. Concerning the attraction of those topics, the chance of eventually taking my doctoral degree based on results achieved as part of (or closely related with) my (research) work may very well have played an important role; after all, a scientific career can hardly be done without a doctoral degree. It therefore gave me much satisfaction when Prof.dr. Emile H.J. Aarts adopted me as a Ph.D. student in January 2003, based on the results achieved in the preceding years.

Since April 1999, I have been employed at the Philips Research Laboratories in Eindhoven and worked in the area of Quality-of-Service for consumer devices, such as digital TV sets and set-top boxes, with a focus on dynamic resource management. Having worked as a software architect in the telecommunications domain in a development organization in the preceding years, it was a challenge to start in an entirely new domain, and a great opportunity to perform research myself. When I started at Philips Research, I became a project member of the Quality-of-Service Resource Management (QoS RM) project. That project was in its definition phase, was meant to become the flagship of the Information Processing Architectures group of the Information and Software Technology sector of Philips Research, and received the special assignment of starting a stream of patents and papers. Due to the pressing needs within that project, I also soon became projectleader, which was a blessing and turned out to be a curse as well.

It was clearly a challenge to set up a company research project in an exciting new domain and to establish contacts and set up co-operations with experts from different domains, both within Philips in the context of the so-called Video-Quality-of-Service (V-QoS) program, as well as outside Philips, within the context of European projects, such as ITEA[1]/EUROPA[2], amongst others. Moreover, it was an honor to be entrusted the co-responsibility of guiding the V-QoS program (together with Christian Hentschel) in general, and the responsibility of guiding the QoS RM project in particular.

Conform the agreements made during my application for a job at Philips Research, I reserved time for my own research. To perform research that would allow me to take a doctoral degree and that could be done next to being a project leader, I carefully selected a topic that fitted in the project, would not be on the critical path of the project, matched my individual strength, and could be done in relative isolation.

Due to the combination of my responsibilities as project leader, my aspirations to perform research myself, and complications surrounding both the birth of my son Wander in August 2001 and his subsequent development, I became overloaded during the autumn of 2001 and dropped out in March 2002. I spent several months with just my family and friends, and needed a year to recover. That eventful year provided me with ample opportunities for reflection and reconsideration of priorities, and changed my life irreversibly.

During my illness, management decided that although the topic of my research was covered by the so-called long range technical objectives of the sector, it was not of sufficient interest to the company to justify its further incorporation in the research program of the group. However, given on the one hand the progress already made (80% of the work towards a Ph.D. thesis was already completed according to the adepts) and on the other hand my aspiration for a scientific career, I was determined and persisted to finish that work. This placed me for an interesting dilemma. In order to complete the work in a surveyable time and to be able to perform similar work in the future, I had to leave Philips and apply for a position at, for example, a university. However, in order to qualify for a position at a university, such as the TU/e, I should be in possession of a doctoral degree. I am grateful to both Philips and the TU/e for granting me the opportunity to complete my thesis by means of a secondment for a period of a year. Assuming successful, I will subsequently make a transfer to the academic world, which allows me to stay at the edge of the

[1]ITEA is an acronym of Information Technology for European Advancement.

[2]EUROPA is an acronym of End-User Resident Open Platform Architecture.

unexplored and continue to perform similar work.

In retrospect, I consider it worthwhile to describe the experience gained with completing the work during my secondment in general, and the spin-offs of that work in particular. During my secondment, a big gap between my analytical results and the existing engineering practice at Philips Research became apparent, and I spent a considerable amount of time closing that gap. The analysis revealed ample opportunities to improve the existing practice, and turned out to have interesting spin-offs from an intellectual property perspective as well; see Table 10.1 on page 187.

Acknowledgement

This thesis could not have been written without being granted the opportunity of a secondment from Philips to the TU/e, and I am greatly indebted to the following persons for making that happen, amongst others: Emile H.J. Aarts, my first promoter, Jaap van der Heijden, head (a.i.) of the former sector Information and Software Technology of Philips Research, Jean H.A. Gelissen, head of the group Information Processing Architectures of Philips Research, Johan J. Lukkien, head of the System Architecture and Networking group of the TU/e, and Suzanne Udo, head of the Mathematics and Computer Science department of the TU/e. Moreover, I thank Philips Research and the TU/e for funding the printing costs of this thesis. The work reported upon in this thesis has been partially funded by the ITEA/EUROPA project [Gelissen, 2001].

Many people have supported my work or contributed to the results presented in this thesis, and it is a privilege to express my gratitude in print.

First of all, I would like to thank my co-promoter Wim F.J. Verhaegh. He accepted the role of supervisor as a matter of course long before the certainty of being allowed to complete this thesis even came in sight. His warm and unique way of supporting my research has given me the strength to continue and complete this research. Moreover, he contributed to a considerable extent to this thesis. In particular, various mathematical proofs would not have their current grace and rigor without his support, and some lemmas and proofs would either not have existed in their current form or not at all. He was always willing to discuss subjects, and read every single letter I wrote.

I am very grateful to Emile H.J. Aarts for adopting me as a Ph.D. student, and for his confidence in and support of my work, especially during the times my work was under fire within Philips Research. Moreover, he played a vital role concerning my secondment. Emile also prevented that I got astray in details and always brought me back to basic ideas. It was both a pleasure and an honor to have him as my promoter.

I am grateful to Jean H.A. Gelissen for his continuous support, especially during the eventful period after I dropped out, and the exceptional situation that subsequently arose. He was simply there when I needed him most. Peter D.V. van der Stok coached me during the period of time I was located at the premises of Philips Research. Peter convinced me to provide a proper model for the schedulability analysis complementing rate monotonic analysis [Bril & Verhaegh, 2003] and suggested the usage of intervals. Moreover, he encouraged me to search for a promoter and to write a thesis. He also pointed me at a vacancy at the TU/e, which became an important element of the secondment.

I am also grateful to Christian Hentschel. We managed and carried the responsibility of the V-QoS program together and closely co-operated. That co-operation was not only motivating and inspiring, but also very pleasant. Christian taught me how to write patents, and alternative ways to write papers and prepare presentations. Moreover, I thank Christian for his constructive comments, support, and above all his friendship.

I thank Gerhard Fohler, my second promoter, for his encouragements, and feedback, for all valuable suggestions for improvements of my thesis, and for introducing me into the real-time community. Johan J. Lukkien provided a pleasant working environment in which I could complete my work in peace and quiet. Moreover, I thank him for the stimulating discussions concerning my research.

As mentioned before, the work described in this thesis has been initiated within the context of the V-QoS program, being a close co-operation between the scalable video algorithm (SVA) project and the QoS RM project. I thank all team-members of that multi-disciplinary program for their co-operation. From Philips Research, these are Christian Hentschel, Maria Gabrani, Ralph A.C. Braspenning, Rob H.M. Wubben, Yingwei Chen, Tse-Hua Lan, Zhun Zhong, C. (Kees) C.A.M. van Zon, Wim F.J. Verhaegh, Clemens C. Wüst, Martin H. Klompstra, Elisabeth F.M. Steffens, Rogier Lodder, Herman Budde, Clara M. Otero Pérez, and Laurenţiu M. Pãpãlãu. I also thank the team-members from the Departamento de Ingeniería de sistemas Telemáticos/Universidad Politécnica de Madrid, Alejandro Alonso, Marisol García Valls, Angel M. Groba, and Jose F. Ruiz. Moreover, I thank C. (Kees) Gravendeel and Hannie C.P. van Iersel from Philips Intellectual Property and Standards organization.

My special thanks go to the spiritual co-parents of this work. The original seed for conditionally guaranteed budgets (CGBs) was planted by Kees van Zon during a meeting in December 1999. The underlying idea of weak CGBs subsequently emerged from discussions with Liesbeth, Alejandro, and Angel. Clara conceived an initial 'in-the-place-of' design for weak CGBs [Otero Pérez, 2001], which gave rise to the challenge for best-case analysis next to worst-case analysis, and provided the basis for an instantaneous budget configuration change as facilitated by CGBs.

Liesbeth came out with and offered me the idea of *restrained* budget use, being one of the founding pillars of strong CGBs. Clemens his work on QoS control strategies for high-quality video processing provided the inspiration for the mechanism to explicitly claim and release a budget margin, being a second pillar for strong CGBs. Wim conceived the so-called *cognac-glass algorithm* to determine the amount of budget that can be conditionally guaranteed based on a budget margin.

It was both a pleasure and an honor to discuss my work with Lui R. Sha, Alan Burns, Sanjoy K. Baruah, and Giuseppe Lipari, and I thank them for their feedback and suggestions.

As mentioned before, I started in an entirely new domain five years ago, and carried the (co-)responsibility of the V-QoS program in general and the QoS RM project in particular for three years. As a result, I did not build up thorough knowledge or experience in the area of real-time systems, and therefore still consider myself to be a novice. It is therefore very likely that I misinterpreted valuable feedback and suggestions or conveyed them erroneously. Any remaining error or obscurity are therefore for the sole responsibility of and claimed by the undersigned.

During the eventful period after my drop out, I was comforted by many people, and I am very grateful for their countenance. From all those people, I would like to explicitly mention Lui R. Sha and Gerhard Fohler, my colleagues Christian Hentschel, Wim F.J. Verhaegh, Peter D.V. van der Stok, Jean H.A. Gelissen, and Clemens C. Wüst, my dear friends Fred Posthumus and Harrie A.C. Tilmans, my caring neighbors Gabriëlle E.E. Schulten, Peter H.G. Pladet, Thea H. Huinen-Fugli, Jacques L.J. Huinen, and Annemiek M.L.J. van Nijnanten, and above all my family.

Finally, I would like to thank all people I have not mentioned before, but who also supported my work and life in a spiritual or technical way.

Eindhoven, July 2004 Reinder J. Bril

Contents

1 Introduction

This thesis is concerned with real-time scheduling for media processing in software in high volume electronics (HVE) multimedia consumer terminals (MCTs), such as digital TV sets, digitally enhanced analog TV sets, and set-top boxes (STBs). Media processing in software is performed using powerful programmable components. In order to be competitive with dedicated hardware solutions, these programmable components have to be used very cost-effectively. Moreover, software solutions should preserve existing qualities of these systems, such as robustness, stability, and predictability. This challenge has been addressed within Philips Research in a so-called Video-Quality-of-Service (V-QoS) program, and the work presented in this thesis has been initiated within that context. In this chapter, we describe the approach taken in that program. It is shown that the approach gives rise to a problem related to user focus. To resolve this problem, this thesis presents the novel concept of a conditionally guaranteed budget (CGB), its design and accompanying implementation, and complements it with novel and essential means to analyze systems with CGBs.

1.1 Multimedia consumer terminals

HVE consumer terminals (CTs) are gradually evolving from straightforward terminals of a video broadcast network (TV sets) and a communication network (telephones) to interactive multimedia terminals, and beyond that to elements in an in-home network, or even an ambient intelligent environment [Aarts, Harwig & Schuurmans, 2001]. In this section, we describe the evolution of CTs to *multimedia* consumer terminals (MCTs), and the trend from analog systems to digital systems, and beyond that to systems where significant parts of the media processing are expected to move from dedicated hardware to software.

In the past four decades, mid-range and high-end TV sets have evolved from stand-alone, analog hardware systems to digital systems containing megabytes of embedded software. Digitization of television started with, and to a large extent still builds on, the digital enhancement of analog standards (e.g. 100 Hz TV). More recently, the emergence of digital video broadcasting (DVB) gave rise to fully digital TV sets, and digital STBs that can be combined with existing analog sets. The digitization of television has clear technical benefits, such as better image and audio quality, more channels, and multi-window TV at moderate additional costs. Most of the digital signal processing is still performed in hardware, however. The embedded software in present-day TV sets is mainly in the area of control and services. Control tasks determine which hardware devices will be on the signal processing path, and what their settings will be, based on characteristics of the set and the incoming signal, and, of course, on user inputs. Typical software-based services include teletext, on-screen display, and menus. When combined with other devices, such as a telephone, game console, or a video recorder, a TV set may provide additional services, such as TV-commerce and personalized teletext (with a telephone), games (with a game-console), and delayed viewing (with a video recorder). Figure 1.1 shows an example of an MCT for high-quality video applications. The MCT may accept input from different types of input sources, and may have a number of video outputs.

Recent developments aim at new kinds of interactive services that drastically change the traditional role of a TV set. It is expected that a significant part of the interactive multimedia services that have evolved on the Internet will be transacted over the television in the near future [Brunheroto et al., 2000]. In addition, the TV set (or STB) is expected to become a multimedia platform, which will serve as a gateway to the Internet [Gran & Scheller, 2000], and may be integrated with other consumer devices within a home network. New architectures for STBs [Lonczewski & Jaeger, 2000] and digital TV sets [Vuorimaa, 2000] explicitly address support for interactive services. As an example, the ITEA[1] project EUROPA[2] [Gelissen, 2001] aimed at defining an STB reference architecture based on the DVB-MHP (multimedia home platform) model, with extended functionality to enable

[1]ITEA is an acronym of Information Technology for European Advancement.
[2]EUROPA is an acronym of End-User Resident Open Platform Architecture.

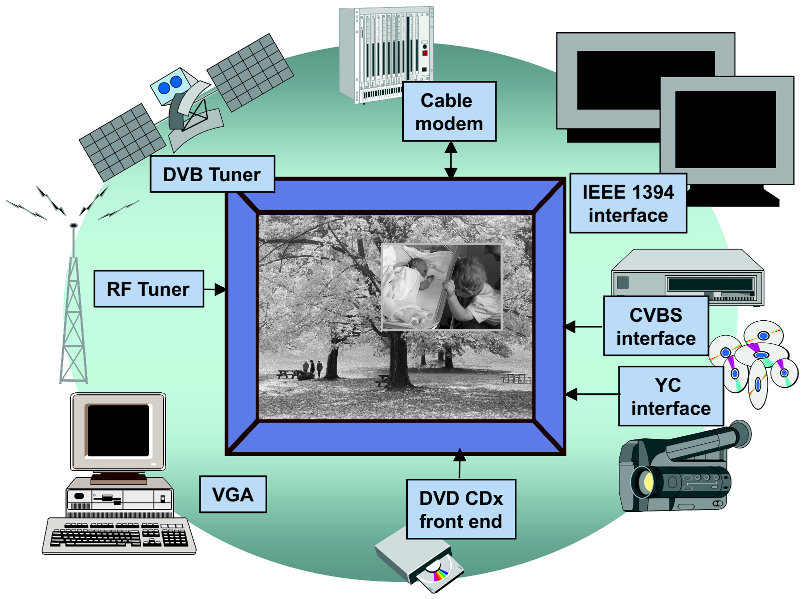


Figure 1.1. Multimedia consumer terminals for high-quality video applications (by courtesy of Maria Gabrani).

next-generation services, such as secure online shopping and banking, based on multi-modal interaction (MPEG-4, MPEG-21), and secure transactions (cryptography). The project closely co-operated with standardization bodies (MPEG, DVB), and is expected to contribute to the conception of a joint (European) STB architecture that will strengthen the position of the European CE and IT industry. The new interactive services require that STBs and TV sets, or with a general term multimedia consumer terminals (MCTs), become open and flexible, not only in the area of control and services, but also in the area of media processing. Since openness and flexibility are typical characteristics of software-based systems, significant parts of the media processing are expected to move from dedicated hardware implementations to software implementations on programmable platforms. However, software media processing is currently more expensive than equivalent processing on dedicated hardware, and more error-prone. Fortunately, *perception* is key for HVE MCTs, and we can therefore fruitfully apply the notion of Quality-of-Service (QoS), as defined in [ITU-T, 1994].

> “The collective effect of service performance which determine the degree of satisfaction of a *user* of the *service*.”

System architects can exploit the QoS concept to design systems that make run-time trade-offs between delivered QoS and consumed resources. Over time, robustness has become a basic requirement for CTs. No one accepts a TV to stall with the message "please reboot the system". To meet the new challenges (*openness* and *flexibility*) and the existing ones (*cost-effectiveness* and *robustness*), future architectures must provide explicitly designed dynamic behavior, explicit management of the available resources (including QoS), and stability under stress and fault conditions.

In the future, the consumer electronics (CE) industry may gradually move in the direction of ambient intelligence. That concept foresees a world in which the CE devices are integrated into the background of people's environment (walls, clothing), and in which it is possible for any person to have access to any source of information, communication and entertainment at any place and any time [Aarts, Harwig & Schuurmans, 2001]. The CE devices that play a crucial role in the ambient intelligence environment will be based on product platform architectures, like the ones that are currently being developed for MCTs.

1.2 Media processing in software

Media processing in software enables MCTs to become open and flexible, and opens the way to use dynamically scalable media functions. Expected advantages over dedicated single-function hardware solutions include versatile, future proof, upgradable products, reduced time-to-market for new features, and reuse of hardware and software modules to support product families. In this section, we will illustrate the resulting business opportunities, briefly discuss the challenges, and outline the co-operative QoS approach taken in the Video-Quality-of-Service (V-QoS) research program within Philips Research.

1.2.1 Business opportunities

We will illustrate the business opportunities of media processing in software in general and scalable media functions in particular by showing how they support product families [Bril, Gabrani, Hentschel, Van Loo & Steffens, 2001] and how they can overcome the inherent constraints of platforms for traditional systems [Hentschel, Bril & Chen, 2002].

Figure 1.2 shows a programmable product family versus algorithm requirements. In this figure, the height of a platform illustrates the available resource capacity, and the height of an algorithm illustrates the resource capacity needed for operation. Programmable platforms with different resource capacities will exist in parallel to suit different market segments. Current media processing algorithms are designed for highest quality at given resource capacity, based on a worst-case complexity of the media data to be processed. Their resource requirements and output quality are usually fixed; see the left column of algorithms in Figure 1.2.

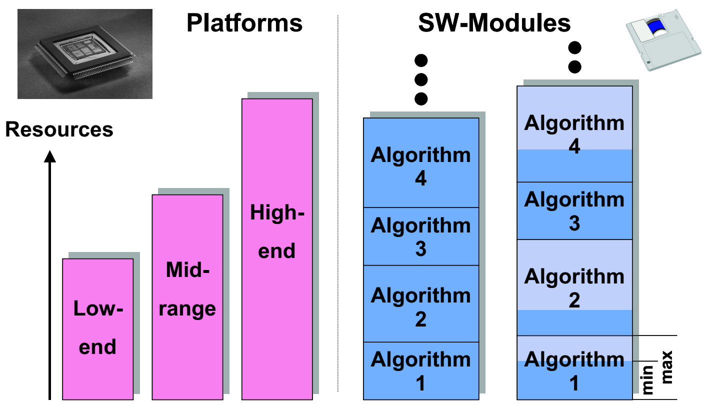


Figure 1.2. A programmable product family, and fixed and scalable software algorithms.

Hence, the number of algorithms allowed to run in parallel is platform dependent and very limited. As an example, only a subset of the algorithms can run on a low-end platform simultaneously. A way to get beyond these limitations is to design scalable video algorithms (SVAs). These may have a kernel, that is not scalable (dark areas), and a part that is scalable to increase the quality of the output (light areas); see the right column of algorithms in Figure 1.2. In this way larger subsets of algorithms can simultaneously run on a low-end platform, albeit at lower quality.

Traditional systems are designed for a specific target functionality at a high quality; see Figure 1.3. Going slightly beyond that target functionality increases

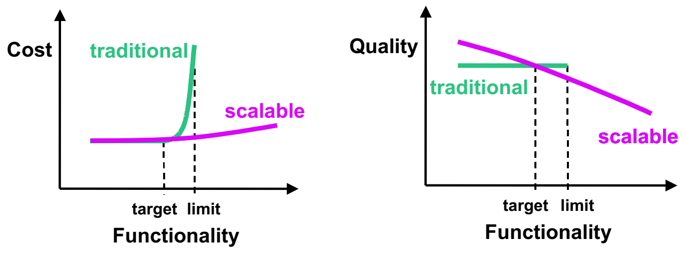


Figure 1.3. Expected cost-effectiveness and quality trade-off of traditional systems compared with a scalable approach.

the cost of development or the bill of material significantly. As an example, the memory or CPU needs of additional functionality may require either an expensive optimization of the code in order to fit the available resources or an upgrade of the platform. An approach based on scalable media functions has no such limits. Instead, given the available resources, the quality depends on the functionality used at a given time. Even an upgrade of existing or later installed applications may run at a very high quality. However, when multiple or complex applications are running concurrently, the quality may drop depending on the available resources.

1.2.2 Challenges

Given their high volume, MCTs require a low bill of material, and these systems must therefore be *cost-effective*. However, compared to dedicated hardware solutions, software solutions are currently expensive in terms of both silicon area as well as power consumption. Cost-effective media processing in software therefore requires a high average resource utilization. Because the worst-case and average-case resource needs of high-quality audio and video applications are typically far apart, this leads to *close-to-average-case resource allocation* to applications. The resulting system will therefore be heavily loaded, and run a high risk of becoming unstable. Trashing in virtual memory systems is a well know example of unstable behavior.

The load of an MCT varies dynamically, and on multiple time-scales. The two main sources of load changes of an MCT are the user and the media data. For a TV set, examples of the former include starting applications, such as switching from one channel to another, and video conferencing when a call arrives. Examples of the latter include changes of the characteristics of the media data, e.g. due to the interruption of a movie (24 Hz film) by a commercial (50 Hz camera) initiated by a service provider, and changing computational requirements, e.g. structural load changes due to motion or shot changes and temporal load changes that happen continuously. Figure 1.4 shows the decoding times for a sequence of MPEG-2 frames on a Trimedia 1300 (180 MHz) processor [Slavenburg, Rathnam & Dijkstra, 1996], illustrating both temporal and structural load fluctuations. The time-scale of these load changes range from minutes for application changes (start/stop) to tens of milliseconds for temporal load changes in computational requirements.

An application running on an MCT can be in a number of different modes. Each *application mode* is determined by a set of *mode parameters* [Bril, Steffens, Van Loo, Gabrani & Hentschel, 2001; Gabrani, Hentschel, Steffens & Bril, 2001]. Very often, mode parameters are characteristics of the media data. A typical mode parameter is the video resolution available at the input, e.g. standard definition (SD) or high definition (HD). These application modes change dynamically, e.g. due to switching from one source or channel to another, and such a change may cause major changes in the resource requirements and clear changes in output quality.

High-quality audio and video have stringent *real-time requirements*. As an example, when a video processing application misses a deadline, a picture may not become available in time, and can therefore not be presented in time. In such a situation, the system will re-display the previous, i.e. ‘wrong’, picture. Deadline

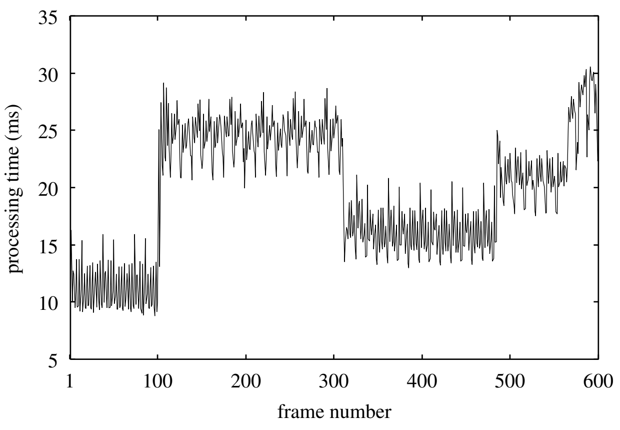


Figure 1.4. Decoding times for a sequence of MPEG-2 frames showing both temporal and structural load fluctuations.

misses tend to come in bursts during periods of heavy load, and without precautionary measures, valuable work may be lost, which is not cost-effective. Moreover, because deadline misses are perceived as non-quality by a user, display of the 'right' picture that is processed at a lower quality level is often preferred. Processing video data at a lower quality when resource needs exceed the allocated resources is a QoS issue, and we may therefore view real-time behavior as a QoS parameter.

As mentioned in the previous section, QoS is determined by the degree of user satisfaction, which has to do with *perception* for high-quality video. As a consequence, only video specialists can make the necessary trade-offs. This will be illustrated by a number of examples. As mentioned above, the perceived quality of a picture with a lower quality is often higher than that of a missed deadline. Similarly, a stream of pictures of a constant lower quality typically has a higher perceived quality than a stream of a slightly higher average quality but with quality fluctuations. Moreover, upon a scene or shot change, the human brain typically needs some time to adjust. A video application may take advantage of this characteristic by temporarily dropping the quality level, e.g. to alleviate overload problems, without the user's notice [Wubben & Hentschel, 2003]. Finally, most people focus on one thing at a time (*user focus*), and that focus is normally at the center of the screen. The output of an application with user focus is evaluated differently by a user than the output of other applications, and thus the application with user focus should be treated differently with respect to quality. Hence, user focus induces a *relative importance* of the applications on MCTs. Stable output quality is a primary quality requirement for the application with user focus.

Replacing dedicated single-function components by *shared* programmable media processing components also introduces the inherent problems of programmable platforms in the area of high-quality media processing. In particular, sharing of resources is a potential source of interference, not only between media applications, but also in combination with the overhead involved in control and interrupt processing. Without precautionary measures, this interference leads to unpredictability, and jeopardizes overall system robustness.

1.2.3 A co-operative QoS approach

Media processing in software for MCTs is required to be cost-effective, while preserving typical qualities of HVE devices, such as robustness, predictability, and stability. Cost-effectiveness requires that the resources are allocated, provided, and used effectively, towards the goal of maximizing the overall QoS. Within Philips Research, this challenge has been addressed by a co-operative QoS approach involving specialists with complementary expertise and responsibilities [Bril, Gabrani, Hentschel, Van Loo & Steffens, 2001]. An overview of the approach is given by Bril, Hentschel, Steffens, Gabrani, Van Loo & Gelissen [2001], and an extensive presentation is given by Steffens [2002]. The approach combines *application adaptation* and *QoS-based resource management*.

As mentioned above, applications running on an MCT can be in a number of different modes. Adaptive applications can operate at different quality levels within an application mode, allowing run-time trade-offs between output quality and resource usage by controlling the *operational quality*. A *resource estimate* is associated with each quality level in each application mode. The applications are responsible for both effective and efficient resource usage for media processing. Examples of adaptive applications from the video domain are given by Hentschel, Braspenning & Gabrani [2001] and from the 3D graphics domain by Lafruit, Nachtergale, Denolf & Bormans [2000].

The basis of the QoS approach is constituted by a software framework for QoS-based resource management, that has been developed in a combined research effort by Philips Research and the Departamento de Ingeniería de sistemas Telemáticos/Universidad Politécnica de Madrid (DiT/UPM) [Otero Pérez, Steffens, Van der Stok, Van Loo, Alonso, Ruiz, Bril & García Valls, 2003]. That framework has been designed as a combination of a *multi-layer control hierarchy* and a *reservation-based resource manager*.

The control hierarchy is responsible for effective, dynamically adjusted, resource allocation, and for effective use of the allocated resources. Overall effectiveness is to a large extent realized by dynamically maximizing the overall QoS, using a general (semantically neutral) notion of *utility*. The optimization is based on the momentarily available mappings from quality levels to resource needs of the applications, the utilities of the individual applications, and also takes the relative importance of applications into account, using a model similar to the one described by Lee, Lehoczky, Rajkumar & Siewiorek [1999a]. By selecting a quality level for an application, the control hierarchy determines the operational setting at which that application is expected to run. In that respect, our approach differs from traditional approaches where applications are greedy, i.e. try to execute at the highest quality as often as possible [Liu, 2000]. The control hierarchy addresses stability by choosing appropriate time scales for the control layers.

The resource manager is responsible for efficient resource provision. The resource manager addresses robustness and predictability by providing *guaranteed resource budgets*, which are based on *resource reservation*. Resource reservation is a well-known technique in operating systems research [Mercer, Savage & Tokuda, 1994; Rajkumar, Juvva, Molano & Oikawa, 1998] to improve robustness and predictability. Moreover, it is recognized as a basis for QoS management [Lee, Lehoczky, Rajkumar & Siewiorek, 1999b]. It is based on four components: admission control, scheduling, accounting, and enforcement. When combined properly, they provide guaranteed reservations. Resource budgets are allocated to so-called *resource consuming entities* (RCEs), which are active components consisting of one or more tasks. Applications consist of one or more RCEs. The resource manager complements resource provision through budgets by mechanisms for spare-capacity provision. Spare-capacity originates from unused (and reclaimed) reservations (so-called *gain time*) and from time that has not been allocated by means of resource reservation (so-called *slack time*); see also [Audsley, Davis & Burns, 1994].

Currently, only CPU budgets are provided by our resource manager, and only a rudimentary mechanism for spare-capacity provision has been implemented. The resource manager is built on top of a commercial-off-the-shelf real-time operating system (RTOS). The CPU resource reservation is based on the fixed-priority scheduling (FPS) model used by that RTOS, and the admission test is based on rate monotonic analysis (RMA). The inspiration for the implementation of budgets was found in papers written by Audsley, Burns, Richardson & Wellings [1993] and Sprunt, Sha & Lehoczky [1989]. Since we strive for a processor utilization that exceeds existing utilization bounds as derived by Liu & Layland [1973] and Bini, Buttazzo & Buttazzo [2001], the implementation of our admission test is based on an exact schedulability test as described by Joseph & Pandya [1986] and Audsley, Burns, Richardson & Wellings [1991]. The admission test assumes a basic real-time scheduling model, as described by Liu & Layland [1973]. A CPU resource reservation based on earliest deadline first (EDF) is currently under investigation.

The guaranteed CPU budgets provide temporal isolation between applications. However, in order to obtain robustness, the applications have to contribute as well. Due to the close-to-average-case resource allocation, and given the dynamic load fluctuations as depicted in Figure 1.4, applications will be faced with temporal (or stochastic) and structural (or systematic) overloads. The resulting robustness problems have to be resolved by the applications themselves. Stated in other words, the applications have to *get by* with their budget. To this end, an adaptive application trades output quality and resource usage, by controlling its operational quality level. Such application-specific control resides at the lowest layer of the control hierarchy. Wüst, Steffens, Bril & Verhaegh [2004] describe QoS control strategies for high-quality video processing. They assume a single-threaded, soft real-time, asynchronous video processing task, that works ahead to even out the load [Sha, Lehoczky & Rajkumar, 1986]. Lan, Chen & Zhong [2001] and Lafruit, Nachtergale, Denolf & Bormans [2000] describe methods to regulate varying computational load for high-quality video decoding and for 3D decoding and rendering, respectively, assuming synchronous processing. A structural overload of an application may require a re-optimization at a higher layer of the control hierarchy.

QoS-based resource management is semantically neutral, and the interface between the applications and QoS-based resource management is therefore also defined in semantically neutral terms. The notions of *utility*, *quality level*, and *RCE* are examples of such terms. Having a unified QoS measure for QoS-based resource management, such as utility, allows the integration of applications of different domains into a single system. On the other hand, the applications may use application-domain specific measures for QoS adaptation. Obviously, specific requirements of applications may result in dedicated mechanisms provided by QoS resource management, but these mechanisms are not application-domain specific and are sufficiently general to be applicable in other contexts as well. This is illustrated by the conditionally guaranteed budgets (CGBs) presented in this thesis. CGBs are implemented as a general mechanism at the level of the resource manager. Although CGBs have been conceived to solve a user-focus problem, they can also be applied in completely different contexts, as will be shown in this thesis.

The basic approach has been successfully applied to single processor systems for scalable video [Hentschel, Bril & Chen, 2001; Hentschel, Bril, Chen, Braspenning & Lan, 2003] and scalable 3D graphics [Van Raemdonck, Lafruit, Steffens, Otero Pérez & Bril, 2002], and it is believed that extensions can be applied to distributed systems.

Each of the basic ingredients of the approach is known, and the same holds for the combination of some ingredients. As examples, the combination of application adaptation and resource reservation has also been described in [Foster, Roy & Sander, 2000; Hamann, Löser, Reuther, Schönberg, Wolter & Härtig, 2001], and hierarchical control for multimedia may already be found in [Uykan, 1997]. However, it is believed that we were among the first to combine these ingredients into a single approach in general and for high-quality video processing in particular.

1.3 Informal problem statement

Multi-layer control combined with guaranteed CPU budgets, and close-to-average-case resource allocation gives rise to a problem related with user focus. Because structural load changes and temporal load changes have different time scales, they are addressed at different layers in the control hierarchy. Upon a structural load increase of an application and without precautionary measures, that application has to face the overload. To get by with its budget, the application will decrease its operational quality level, and the quality of its output will therefore decrease. For stability reasons, the control layer that addresses structural load changes is triggered when it is sufficiently certain that the overload was indeed the result of a structural load increase. The trigger may subsequently result in a re-allocation of budgets, and a request to the resource manager to perform a so-called *budget configuration change* (BCC). When the application that suffered the load increase had user focus, the system may aim at restoring its output quality at the cost of the output quality of other applications. In summary, upon a structural load increase of an application with user focus, its output quality has a dip. In the remainder of this document, the problem of the temporary quality dip in the output of the user focus application will be referred to as the *user-focus problem*.

Below, we will first describe the notion of user focus and the user-focus problem in more detail. Next, we show that the user-focus problem is a specialization of a more general problem. Although we primarily aim at resolving the user-focus problem in this thesis, we are inclined to believe that our solution equally well applies to its generalization. The section is concluded with a sketch of our solution.

1.3.1 User focus

A TV set may support a variable number of windows, such as a main window (showing, for example, a movie) and one or more secondary windows, e.g. PiP, videophone, or a web-browser. The user's focus is (typically) on one thing at a time, but changes dynamically from one window to another. Windows having user focus are evaluated differently by a user than other windows, and thus the applications with user focus should be treated differently with respect to quality. Hence, user focus induces a *relative importance* of the applications of consumer terminals. This relative importance is taken into account during the overall system optimization, and a change of user focus requires a re-optimization. Stable output quality is a primary quality requirement for the application with user focus.

1.3.2 User-focus problem

Consider two applications that are running in a fully loaded system. The outputs of both applications are visible to the user. The output of one application (identified by UF) has user focus, whereas the output of the other application (identified by ¬UF) does not have user focus. In addition to these two applications, there may be other, so-called neutral, applications. For ease of presentation, we restrict ourselves to UF and ¬UF in this section.

Problematic behavior

Figure 1.5 visualizes the user-focus problem by showing the load induced by the input data (lines *a* and *b*) and the perceived output quality of both applications (lines *c* and *d*) as a function of time. When a sudden increase of the induced load of UF occurs (at time t_I), UF faces a structural overload situation due to the average-case resource allocation. The application-specific control of UF may detect the structural overload and subsequently signal the problem to a system controller at a higher layer in the control hierarchy. Next, the system determines the new optimal quality levels at which UF and ¬UF will run. We assume that the quality level for UF remains the same. Thus, after a certain reaction time (from t_I to t_A), a so-called *mode change* is initiated. The quality and the budget of ¬UF are reduced, in this order, and, subsequently, the budget of UF is increased. At time t_S, a new equilibrium is reached. In the mean time (from t_I to t_S), the perceived quality of UF's output is degraded, because UF's resource budget is temporarily insufficient to cope with the increased load, and UF has to get by on its budget, which necessarily results in some form of quality reduction at the output. Thus, the perceived quality at the output of UF is temporarily reduced, even though the set quality level for UF remains the same.

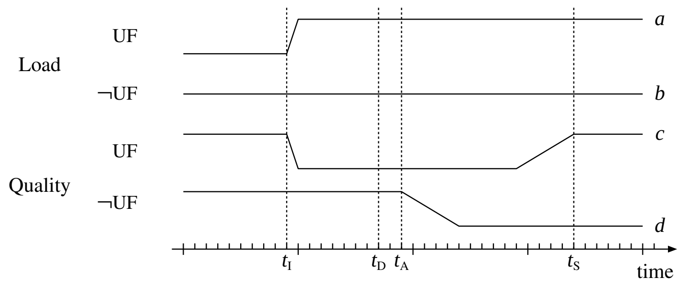


Figure 1.5. Problematic behavior upon a structural load increase of the user-focus application UF.

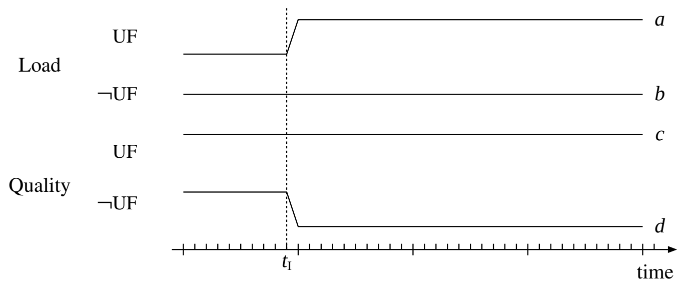


Figure 1.6. Desired behavior upon a structural load increase of the user-focus application UF.

Desired behavior

In Figure 1.6, the desired behavior is shown. UF's perceived output quality is not affected by the sudden load increase, whereas the quality of ¬UF is degraded smoothly. There are two major difficulties in obtaining this desired behavior with average-case resource allocation.

- *Very quick reaction required ⇒ cuts through layers.*

  The desired stable output quality for UF can only be achieved if the additional resources become available *instantaneously*. However, detecting the structural load change (from t_I to t_D), determining the new mode (from t_D to t_A), and effectuating the new mode (from t_A to t_S) takes time. For stability reasons, it is undesirable to react quickly upon a load increase without being sufficiently certain that the overload was indeed the result of a structural load increase. Even when the detection can be done at the moment or before the overload appears (i.e. when $t_D \leq t_I$), a sufficiently quick reaction to support the desired behavior for UF can only be achieved when the new mode is *anticipated* and the BCC can be effectuated *instantaneously*. Hence, the user-focus problem can be traced down to two main complementary causes.

  1. *UF is confronted with a structural resource shortage.*

     After the (structural) load increase, the UF has insufficient resource capacity to meet the requirements of the set quality level. This results in an overload situation in which UF has to get by on its budget, which in turn results in a degradation of UF's perceived output quality.

  2. *The reaction time is too long.*

     In addition to the time needed to detect the structural overload, the system needs time too to determine the new mode and subsequently effectuate it. The total reaction time is too long in the sense that it gives rise to the dip in the quality of the output of UF.

- *Smooth degradation of ¬UF may not be feasible.*
  To achieve a smooth transition for ¬UF, the quality of ¬UF must be reduced first, followed by a reduction of the budget of ¬UF. However, in order to address the needs of UF, the resources allocated to ¬UF will be taken away *instantaneously*, and the quality level of ¬UF will be adjusted after a certain reaction time. In the mean time, ¬UF will have to get by on its remaining resources, and the perceived quality very much depends on ¬UF's ability to do so. In particular situations, the behavior shown in Figure 1.6 may not be feasible, and an overshoot of the perceived quality reduction of ¬UF cannot be prevented; see Figure 1.7. The behavior as exhibited in Figure 1.7 is nevertheless preferred above that of Figure 1.5, because a dip in the quality of the output of ¬UF is presumed to be less problematic than a dip in the quality of the output of UF.

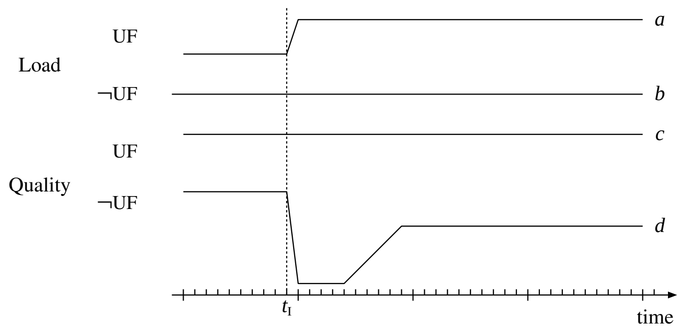


Figure 1.7. Quality degradation with overshoot for ¬UF (line *d*).

1.3.3 A generalization

The distinguishing characteristics of the problem described in this section are the conflicting requirements of the need for a stable output quality of an application in the presence of a structural load increase on the one hand and of a close-to-average resource allocation for cost-effectiveness reasons on the other hand. The notion of user focus specializes this general problem towards a problem of maintaining a stable output for an application with user focus at the cost of the perceived quality of another (visible) application without user focus. Obviously, visibility of the outputs of applications is not an issue for the general problem. As an example, it may be more important to maintain a stable output quality of an application that records video on disk than to maintain a stable output quality of an application providing visible, but highly volatile, output on a screen. Rather than using UF and ¬UF, we will therefore use the more general terms MIA and LIA to denote a *more* *important application* and a *less important application*, respectively. Although the term user-focus problem is too specific for the general problem addressed in this thesis, we will remain using this term for ease of presentation and for historical reasons.

1.3.4 A sketch of our solution

In essence, our solution *anticipates* a structural load increase for MIA. To this end, MIA receives an additional budget, a so-called *budget margin*, to accommodate the load increase. This budget margin of MIA can be accommodated by giving LIA a lower budget. To gain back on the cost-effectiveness loss and to compensate LIA for its budget loss, LIA receives an additional budget with a *conditional guarantee*. This so-called *conditionally guaranteed budget* (CGB) is available to LIA when MIA does not use its budget margin, and is based on that margin. To facilitate an *instantaneous* BCC we present the concept of *in-the-place-of CGB provision*. According to this concept, the CGB is provided to *LIA at* the period and phasing of the budget of *MIA* and *when* that margin would have been provided to *MIA*. In order to determine the worst-case amount of CGB that can be conditionally guaranteed to LIA, we therefore need to know when that margin becomes available. To this end, we need the novel notion of *occupied times*, amongst others. Our solution is explained in more detail in Chapter 7, and the corresponding analysis is the topic of Chapter 8.

1.4 Contributions

To resolve this user-focus problem, we refine the resource reservation technique as currently supported by existing resource kernels. To this end, we present the novel concept of a *conditionally* guaranteed budget (CGB). Unlike a normal budget, which has an absolute guarantee, a CGB can only be allocated with a *conditional guarantee* based on a *conditional* admission test. To distinguish a normal budget from a CGB, we sometimes use the term *absolutely guaranteed budget* (AGB). CGBs are a *mechanism* at the level of the resource manager. Because the resource reservation of our existing resource manager is based on an FPS model, so is the implementation of our CGBs, and we therefore present a conditional admission test based on RMA. For this admission test, we need various extensions to RMA. In the remainder of this section, we describe the two main topics of this thesis, CGBs and the extensions to RMA, in more detail. We conclude this section with an overview of our results that have already been published.

1.4.1 Conditionally guaranteed budgets

CGBs are a *mechanism* at the level of the resource manager, that facilitate *instantaneous* BCCs. Combined with load increase *anticipation*, CGBs can be exploited by *policies* in the control hierarchy to improve the cost-effectiveness of the reservation-based system using *controlled* quality improvements. Being a general mechanism, CGBs can be used in other contexts as well.

This thesis presents two main variants of CGBs, *weak* CGBs and *strong* CGBs, where the terms weak and strong refer to the conditional guarantee. Weak CGBs are aimed at situations where a structural load increase can not be detected in time. On the other hand, strong CGBs are aimed at situations with timely detection, in which applications voluntarily restrain their budget use to a budget without a budget margin and explicitly decide to use their budget margin when needed. Moreover, strong CGBs provide support for asynchronous applications to voluntarily restrain their budget use, as explained in Chapter 7. That chapter also presents the concept of *in-the-place-of CGB provision*, that enables an instantaneous BCC, and can be applied for both variants of CGBs. Accompanying implementations for CGBs for the CPU are described as extensions to the existing resource manager. The presentation also covers budget accounting and budget enforcement for CGBs.

We sketch how our design and implementation of CGBs can be complemented with *in-the-place-of gain-time provision*. The inspiration for in-the-place-of gain-time provision was found in [Caccamo, Buttazzo & Sha, 2000].

1.4.2 Extensions to rate monotonic analysis

For the admission test for CGBs, we need various extensions to RMA. In particular, we need *best-case response times* next to worst-case response times, and the novel notion of (best-case and worst-case) *occupied times*. Best-case response times and best-case occupied times are based on a notion of *optimal instant*.

The extensions to RMA can be applied in other contexts as well. In particular, it will be shown how best-case response times can be applied in the context of *jitter analysis*, and how worst-case response times and occupied times derived for fixed-priority preemptive scheduling (FPPS) can be applied to determine exact worst-case response times for fixed-priority scheduling with deferred preemption (FPDS).

Finally, we improve the efficiency of exact admission tests for situations where response times and occupied times are needed. To this end, we analyze the efficiency of determining best-case and worst-case response times and occupied times, propose alternative initial values for the iterative procedures to determine the response times and occupied times, and analyze their merits.

1.4.3 Published results

Various preliminary results of this work have already been published. The user-focus problem and a solution in terms of weak CGBs has been presented by Bril & Steffens [2001]. They also provide a description of a straightforward implementation as an extension to the existing resource manager. An initial description of the in-the-place-of design for CGBs has been given by Otero Pérez, Bril & Steffens [2001]. They also describe a first implementation of that design for weak CGBs, and initial analytical results based on best-case and worst-case analysis, including the need for and an intuitive description of occupied times. Best-case response times have been described by Bril, Steffens & Verhaegh [2001] and Bril, Steffens & Verhaegh [2004], including their application in the context of jitter analysis. A description how worst-case response times and occupied times derived for FPPS can be applied to determine exact worst-case response times for FPDS is presented by Bril, Verhaegh & Lukkien [2004]. Finally, efficient calculations of response times and occupied times and alternative initial values for the iterative procedures are the topic of papers written by Bril, Pol & Verhaegh [2002] and Bril, Verhaegh & Pol [2003].

1.5 Related work

Our work aims at resolving the user-focus problem. The notions of *user focus* and *relative importance* have also been recognized by others, e.g. [Mercer, Savage & Tokuda, 1994; Ott, Michelitsch, Reininger & Welling, 1998; Lee, Lehoczky, Rajkumar & Siewiorek, 1999a]. Huang, Wan & Du [1998] introduced the related notion of *criticality* to capture the semantics of application importance in the context of mission-critical multimedia applications. The period transformation method described by Sha, Lehoczky & Rajkumar [1986] can be used to ensure that the priority of a task is also an accurate statement of its relative importance. That method has been conceived to make RMS applicable to *transient* overloads. The extension of the interpretation of relative importance with load increase anticipation has not been addressed in the literature before.

Our work aims at maintaining a stable output quality for a more important application upon a *structural* load increase, optionally at the cost of reducing the output quality of a less important application, by dynamically changing resource provisions. Conversely, the work reported upon by Wüst & Verhaegh [2004] aims at maximizing the perceived quality for fixed resource budgets.

Our work started from an existing resource manager, the so-called *budget scheduler*. This manager still assumes a basic real-time scheduling model as described by Liu & Layland [1973]. Many restrictions of that basic model have been lifted by subsequent work. Summaries of results are given by Burns [1991], Lehoczky, Sha, Strosnider & Tokuda [1991], and Sha, Klein & Goodenough [1991], and an historical perspective into the development of FPPS (up to the end of 1993) is given by Audsley, Burns, Davis, Tindell & Wellings [1995]. Although the budget scheduler could be enhanced to cover extensions of that basic model, there was no immediate cause justifying such enhancements. What's more, the support for aperiodic and sporadic tasks provided by a previous version of the budget scheduler is not prolonged in the existing budget scheduler. Most of the analysis presented in this thesis is therefore also confined to a basic real-time scheduling model and excludes extensions, such as sporadic tasks [Mok, 1983], arbitrary deadlines [Lehoczky, 1990; Tindell, 1992], aperiodic tasks [Sprunt, Sha & Lehoczky, 1989], task interaction [Sha, Rajkumar & Lehoczky, 1990; Baker, 1991] and blocking [Audsley, Burns, Richardson, Tindell & Wellings, 1993], precedence constraints and tasks with varying priorities [González Harbour, Klein & Lehoczky, 1994; Groba, Alonso, Rodríques & García Valls, 2002], and time-offsets [Tindell, 1994; Palencia & González Harbour, 1998; Mäki-Turja & Nolin, 2004].

In the remainder of this section, we compare the two main topics of our work with work presented in the literature.

1.5.1 Conditionally guaranteed budgets

We compare CGBs with related work along different axes. The first axis is the general approach of AGBs and mechanisms for spare-capacity provision. Lipari & Baruah [2000] present an algorithm to schedule reservations that is able to efficiently reclaim gain time, and is, generally speaking, fair. The reservation model described by Rajkumar et al. [1998] distinguishes three different types of reservations (i.e. hard, firm, and soft) based on the way they compete for spare capacity. Budget sharing for overrun control, as described by Caccamo et al. [2000], is another way to allocate spare capacity. The primary goal of their spare-capacity allocation algorithm is to improve the *robustness* of a system. CGBs are (very) different from AGBs and mechanisms for spare-capacity provision. They differ from AGBs by being inherently conditional as expressed in the (conditional) admission test. They differ from mechanisms for spare-capacity provision by their very nature of being budgets, having an admission test and being enforced. CGBs may be viewed as a refinement of the reservation models currently supported by existing resource kernels.

The second axis is mode change protocols. In existing mode change protocols, such as described by Tindell, Burns & Wellings [1992], an 'old' mode version of an application is allowed to terminate gracefully. Although a desirable property, a smooth degradation of the quality of LIA is not a requirement for a solution to the user-focus problem, and is not always feasible either. The work described by Sha, Rajkumar, Lehoczky & Ramamritham [1989] and Tindell, Burns & Wellings [1992], seeks to guarantee a priori the timing constraints of *all* tasks (or budgets) across the change from one mode to another. Although the scheduling mechanism for CGBs warrants the guarantees of AGBs, the timing constraints of CGBs are only conditionally guaranteed. To accommodate a stable output quality of MIA as the primary requirement, CGBs facilitate *instantaneous* BCCs. Such changes can be accommodated by applying the concept of *in-the-place-of resource provision* for CGB provisioning. However, this can result in the withdrawal of a CGB *in the midst of* its provision upon such a change, as described in Chapter 7.

The third axis is concerned with servers. The resources of existing servers for fixed priority scheduling, such as those described by Sprunt, Sha & Lehoczky [1989], either become available periodically or sporadically. In contrast, a CGB becomes available with (absolute) *jitter* when in-the-place-of CGB provision is applied, as explained in Chapter 7.

The notion of a CGB has not been addressed in the literature before.

1.5.2 Extensions to rate monotonic analysis

Lower bounds for best-case response times have been presented by Palencia Gutiérrez, Gutiérrez García & González Harbour [1998] and Kim, Lee, Shin & Chang [2000], and applied to reduce jitter estimations. Exact best-case response times were conceived independently by Redell & Sanfridson [2002], including their application in the context of jitter analysis. We are inclined to think that the proofs presented in this thesis are more rigorous than those given by Redell & Sanfridson [2002]. In particular, we uniquely arrive at the recursive equation for best-case response times, whereas Redell & Sanfridson [2002] derive two, in our view, essentially different equations and subsequently select the one with the best characteristics. The intuition for *worst-case occupied times* may already be found in a paper by Thuel & Lehoczky [1994]. However, the notion of worst-case occupied time and a dedicated recursive equation with accompanying iterative procedure to determine its value are left implicit in that paper. Best-case occupied times have not been addressed in the literature before. Moreover, we are not aware of any publication showing that the duality of the various best-case and worst-case notions is reflected in the lemmas and theorems and their proofs.

The efficiency of an exact schedulability test has been the topic of many papers, see [Bini & Buttazzo, 2002] for a recent example. Because we need response times and occupied times for our analysis, we studied the efficiency of calculating those values, and proposed alternative initial values for the iterative procedures. Though conceived independently, the alternative initial value to determine worst-case response times was already discovered much earlier and presented by Sjödin & Hansson [1998] and Sjödin [2000]. Though similar in nature, the initial values for best-case response times have not been reported in the literature before. Moreover, application of the initial values to determine occupied times is new. Our analysis of the efficiency of determining response times and occupied times presents novel insights, and refines earlier conclusions drawn in the papers mentioned above.

This thesis provides a conjecture for the novel notion of an ε-critical instant and presents equations to determine exact worst-case response times for FPDS and arbitrary phasing. Worst-case response times for FPDS have been addressed by Burns [1994], Burns & Wellings [1997], Burns [2001], and Lee et al. [1998]. We show that existing approaches are either pessimistic or optimistic, hence not exact.

As mentioned before, we assume a basic real-time scheduling model, and our extensions to RMA therefore only complement basic analysis.

1.6 Thesis outline

This thesis is organized as follows. Chapter 2 elaborates on the context of the work presented in this thesis, by giving a circumstantial description of the QoS approach to media processing in software.

Chapters 3 till 6 are concerned with FPPS. Chapter 3 introduces real-time scheduling and presents our model for FPPS. The notions of *optimal instant*, *best-case response time*, and *best-case start time* are also introduced in this chapter, and shown to be duals of their worst-case equivalents. Worst-case analysis is the topic of Chapter 4. It recapitulates the notion of critical instant and a recursive equation for worst-case response times and an associated iterative procedure to determine them. The notion of worst-case start time is also addressed in this chapter, and its generalization *worst-case occupied time*. Best-case analysis is the topic of Chapter 5. Together, these latter two chapters provide a basic set of recursive equations with associated procedures for analysis. Chapter 6 addresses the efficiency of the iterative procedures to determine response times and occupied times.

Chapters 7 and 8 concern CGBs. Chapter 7 presents the concept of CGBs, and explains how CGBs solve the user-focus problem. Next, BCCs are analyzed, and the concept of in-the-place-of resource provision is presented as a means to accommodate instantaneous BCCs. Challenges for the design and implementation of CGBs are illustrated by means of examples, and various options and alternative solutions are presented as extensions of the existing budget scheduler. Given the variety of different options, this chapter has an explorative character, and is more engineering rather than mathematically oriented. Whereas Chapter 7 addresses the mechanism for scheduling CGBs, Chapter 8 provides the analytical foundation for the corresponding admission test of CGBs based on Chapters 3 till 5.

Chapter 9 presents two further case studies, showing how the techniques presented in Chapters 4 and 5 can be applied to analyze systems with jitter, and to arrive at exact worst-case response times for FPDS.

Finally, we conclude this thesis in Chapter 10.

2

Media Processing in Software

In this chapter, we consider media processing in software for high-volume electronics (HVE) multimedia consumer terminals (MCTs), and elaborate on the cooperative QoS approach taken within the V-QoS program in Philips Research. Apart from elaborating on the context of the work presented in this thesis, this chapter also provides a brief description of the existing resource manager, the so-called *budget scheduler*, which serves as the basis for our implementation of CGBs, and discusses the need for on-line calculations of exact schedulability tests.

We start with a comparison between media processing in present-day CTs and media processing in software in Section 2.1. It is shown in that section how a QoS approach can alleviate the problem of additional costs of media processing in software. Section 2.2 presents a demonstrator that has been built within V-QoS to show the feasibility of the QoS approach. The next sections describe the two main pillars of our approach, being adaptive applications and QoS-based resource management. This chapter is concluded with a brief description of the budget scheduler, in Section 2.5. This latter section includes a discussion on the need for on-line calculations of response times and occupied times as part of the evaluation of an admission test.

2.1 Dedicated components versus programmable components

As mentioned in Chapter 1, CTs are heavily resource-constrained, with a high pressure on silicon cost and power consumption. Compared to dedicated single-function components, programmable components are currently expensive, both in cost and power consumption. In this section we will show that this problem can be alleviated by sharing programmable components between functions, and by making these functions dynamically scalable. Our comparison is kept simple, and only meant for illustration purposes. The interested reader is referred to [Jaspers, 2003] for an in-depth discussion of trade-offs to be made during architectural design of video processing systems.

2.1.1 Dedicated single-function hardware components

Figure 2.1 shows the basic architecture of a high-end TV, with analog TV input, a standard decoder for both PAL and NTSC, and a dedicated IC for picture improvement (PICNIC; see [De Haan, 2000] Section 2.2). Figure 2.2 shows the same

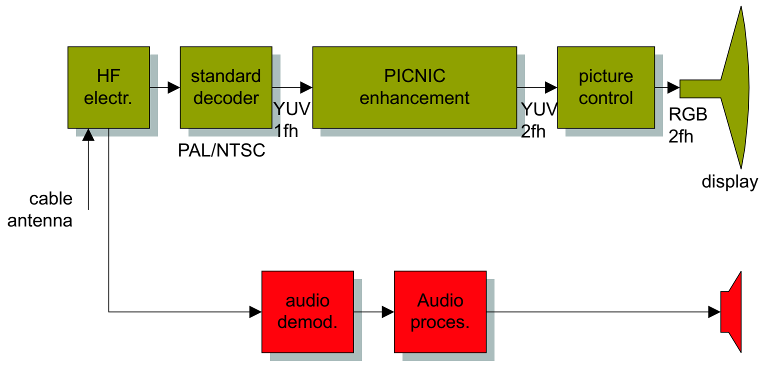


Figure 2.1. Basic hardware architecture of a high-end TV (by courtesy of Egbert G.T. Jaspers).

basic architecture, but extended with components for TXT, picture improvements for PAL (see [Hentschel, 1998] Chapter 2), natural motion (based on the FALCON IC; see [Lippens et al., 1996]), picture-in-picture (PiP) based on analog TV input (CVBS), digital audio (NICAM), and digital TV input (MPEG). Note that each additional feature requires additional, dedicated components.

2.1.2 Programmable components

Rather than requiring additional dedicated, single-function components for each additional feature, media processing in software enables additional features by means of sharing programmable components. In this section, we distinguish

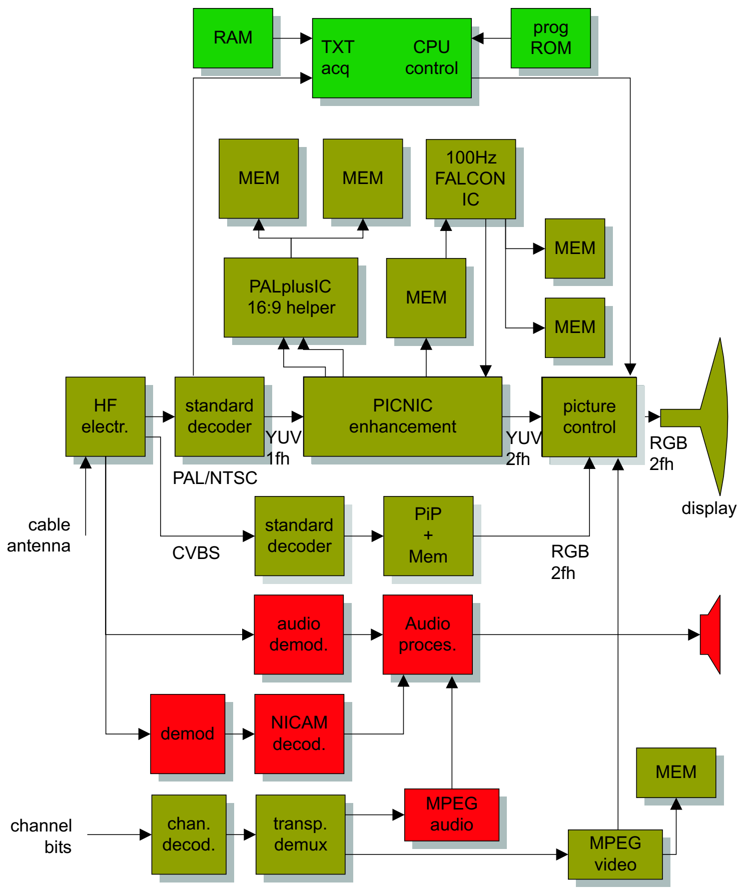


Figure 2.2. Hardware architecture of a high-end TV with extended functionality (by courtesy of Egbert G.T. Jaspers).

three stages to provide more functionality and/or better quality using shared programmable components: *flexibility*, *scalability*, and *QoS*. Software flexibility allows mid-range and low-end products to be in a number of different modes of operation, with different functionality. Scalable media functions extend flexibility by enabling the provision of high-end functionality in mid-range and low-end products, albeit at a lower quality. Scalability may be further enhanced by optimizing the overall perceptual quality at run-time and supporting seamless switching between modes. The term QoS relates to this enhanced form of scalability.

We will consider these three stages in more detail below. This section is concluded by relating these stages to the various kinds of diversity as supported by a software component-based approach.

Flexibility

With flexibility, a system can be in a number of different modes of operation, with different functionality, e.g. NTSC with picture improvement, or ATSC (digital TV input). Reconsidering Figure 1.2, not all fixed software algorithms can be run simultaneously on the low-end platform. Figure 2.3 shows three different modes of operation for the low-end platform, each providing a different subset of high-quality functionality.

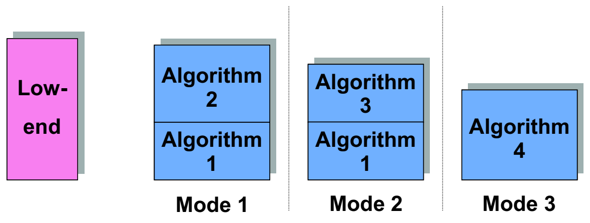


Figure 2.3. Flexibility on a low-end platform, allowing different modes that each provide a different subset of high-quality functionality.

Scalability

Whereas single-function hardware components are generally designed to provide a fixed quality of the output, based on worst-case complexity of the media data to be processed, software functions can be designed to provide different levels of quality, matching the available resources. Scalable media functions make it possible to provide high-end functionality on mid-range and low-end platforms, albeit at a lower quality. With scalability, a system can allow for a combination of functionalities on a platform that normally would not have sufficient resources. As an example, the flexible TV set mentioned above can now show an ATSC input in its main window, and an NTSC input in a small PiP window at the same time, by reducing the resource usage, and hence the quality, for the main window. Given

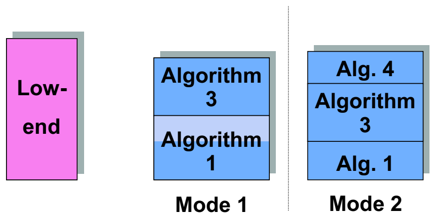


Figure 2.4. Scalability on a low-end platform.

the scalable versions of applications shown at the right-hand side of Figure 1.2, on the low-end platform, for instance, algorithm 3 can be combined with high-quality algorithm 1, or with low quality algorithms 1 and 4; see Figure 2.4.

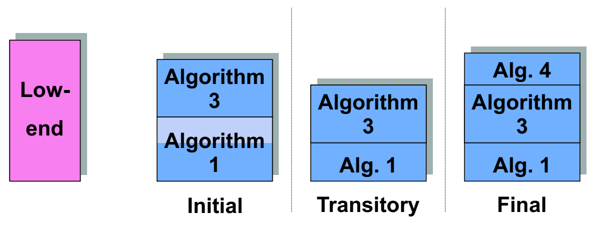


Figure 2.5. QoS on low-end platform.

Quality of Service

QoS enhances scalability, by optimizing the overall QoS of the system at run-time and supporting seamless switching between different modes of operation. In the previous example, when a user opens a PiP window, the selection of the qualities for the main window and the PiP window is optimized towards overall QoS, and the quality of the main window is reduced smoothly to free the necessary resources for the PiP window. This is shown is shown in Figure 2.5. The admission of algorithm 4 is requested in the initial situation. The system first determines the optimal quality levels of the algorithms at which the new mix will be run. The new quality levels are subsequently effectuated. Starting from the initial situation, the quality of algorithm 1 is reduced smoothly. Once the necessary resources for running algorithm 4 at low quality are freed (the transitory situation), algorithm 4 can be started at low quality as a next step (yielding the final situation). When algorithm 4 is terminated at a later moment in time, the available resources may again be allocated to algorithm 1, optimizing QoS and returning to the initial state.

To support seamless switching between modes for QoS, the adaptive applications and the software framework have to complement each other. The applications are responsible for smooth transitions between quality levels and the software framework must ensure that the applications get the necessary resources to do so.

Relation to a software component-based approach

Up to now we considered platforms and applications with no reference to software components. However, by viewing (subsets of) applications as components, we can relate our examples to the various kinds of *diversity* as supported by, for example, Koala [Van Ommering, 1998]. The different modes of operation as supported by flexibility (see Figure 2.3) may be viewed as *structural* diversity. The additional modes of operation as supported by scalability (see Figure 2.4) may be viewed as *internal* diversity of applications 1 and 4. However, the run-time optimization of the overall quality and the run-time requirements posed on the transitions between different modes as supported by QoS (see Figure 2.5) are complementary to a component approach.

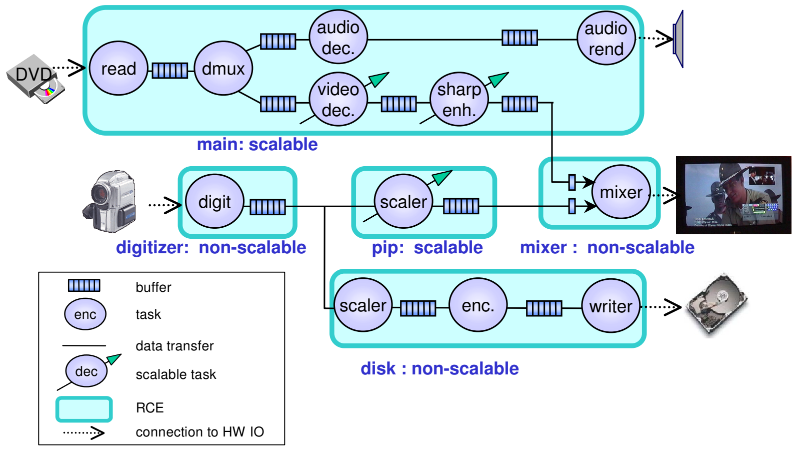


Figure 2.6. Application execution model of the demonstrator.

2.2 Demonstrator

To show the feasibility of our QoS approach and to test the concept, a demonstrator has been built within V-QoS, using a Trimedia 1300 (180 MHz) processor. The demonstrator shows that an application can be added to a fully loaded terminal, and illustrates basic stability during mode changes and robustness upon overloads. We only briefly describe the demonstrator in this section, and refer the interested reader to [Otero Pérez & Nitescu, 2002; Hentschel, Bril, Chen, Braspenning & Lan, 2003] for more information.

2.2.1 Application execution model

Figure 2.6 depicts an application execution model of the demonstrator, taken from [Otero Pérez et al., 2003] and first presented by Hentschel, Bril, Chen, Braspenning & Lan [2002]. Remember that resource budgets are allocated to resource consuming entitys (RCEs). Moreover, an application consists of one or more RCEs, and an RCE consists of one or more tasks. The figure contains five RCEs constituting a set of media applications; three single-RCE user-applications, *main*, *pip*, and *disk*, and two independent RCEs, *mixer* and *digitizer*, that belong to the non-scalable supporting infrastructure. Note that an RCE may contain multiple tasks, and that a task may, but need not, be scalable.

The application *main* takes an MPEG-2 stream with standard resolution (SD, i.e. 720 pixels by 576 lines) as input and provides an audio stream and an SD video stream for a main window as output. The scalable MPEG-2 decoder of *main* is

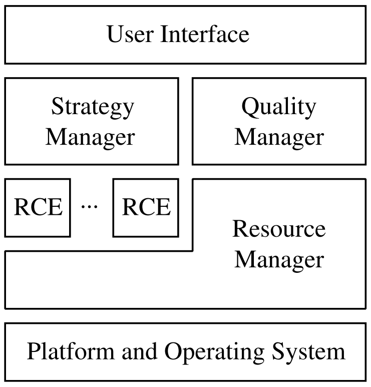


Figure 2.7. Main constituents of the demonstrator.

described by Peng [2001] and Zhong, Chen & Lan [2002], and details on the scalable sharpness enhancement algorithm are presented by Hentschel, Braspenning & Gabrani [2001]. The analog video input stream from the camera, which also has standard resolution, is digitized before being presented to the applications *pip* and *disk*. The application *pip* provides a video stream in QCIF format (i.e. 180 pixels by 144 lines) as output for a PiP window. The application *disk* scales the video input down from SD-format to CIF format (i.e. 360 pixels by 288 lines), and records video in MPEG-1 format on disk.

The demonstrator can run in a number of different modes. A first mode provides only *main*, a second mode provides *main* and *pip*, and a third mode provides all three applications. In both the first and second mode, the scalable algorithms can run at their highest quality levels for most of the time. In the third mode, the applications are only able to provide smooth output when all scalable algorithms are running at their lowest quality levels.

2.2.2 Main constituents

Figure 2.7 depicts the main constituents of the demonstrator. This diagram was a result of discussions between experts from the high-quality video domain and system software specialists, amongst others. A variant of this diagram that also covers 3D graphics has been presented by Van Raemdonck, Lafruit, Steffens, Otero Pérez & Bril [2002].

The application part consists of a strategy manager and a number of RCEs. An application consists of a set of interconnected RCEs. As shown in Figure 2.6, an RCE may contain multiple media processing components. The system part is composed of a resource manager and a quality manager.

The strategy manager and quality manager cooperate to determine the preferred quality settings and the budgets for the RCEs. The strategy manager collects information about resource estimates for all quality levels and provides a strategy for the overall quality optimization of a running application. The strategy manager deals with application domain semantics. The quality manager, on the other hand, works with a general (semantically neutral) notion of *utility*. The quality manager optimizes the system utility based on the utilities of the individual applications, the relative importance of these applications and their momentary available mapping from quality level to resource needs. It also starts and stops applications. For illustration purposes, the demonstrator contains a means to manually override the settings of the quality manager.

In the demonstrator, the quality manager is implemented as a simple optimization function that maximizes the system utility given the quality levels and the resource estimates of the active RCEs. The strategy manager assumes a fixed set of applications, and contains a look-up table that determines the quality settings of the scalable tasks for each mode.

2.2.3 Evaluation

Mode 3 shows that an application (*disk*) can be added to a consumer terminal that is already fully loaded (mode 2), albeit at a lower quality of all applications. The stability of the system was illustrated not only during mode changes but also during normal operation; none of the applications crashed or had to be terminated. Robustness upon overload was illustrated by means of a manual override in mode 3 of the quality setting of the application *main*. Although the visual quality of *main* decreased due to the overload situation and interrupts of the audio were clearly noticeable, *main* was able to get by with its budget. This was realized by default degradation techniques of the MPEG-2 decoder, such as skipping frames for video. The other applications were not affected, illustrating the temporal isolation between applications provided by the CPU budgets.

2.3 Application adaptation

In this section, we consider application adaptation, being one of the two pillars of our co-operative QoS approach. Focus of our V-QoS program has been on the high-quality video domain, and we start with a motivation for that focus in Section 2.3.1. We subsequently describe QoS parameters for high-quality video in Section 2.3.2, and compare them with those from other types of media processing. Section 2.3.3 presents an overview of scalable video processing. Finally, QoS control strategies for high-quality video processing are briefly described in Section 2.3.4.

2.3.1 Focus on high-quality video

The basic media in MCTs are high-quality audio and video. If the basic media processing functions are scalable, other media processing functions can be added at little or no extra cost. Scaling audio is less important than scaling video, for two reasons. Firstly, combined with lower quality audio (e.g. mono), video is perceived at lower quality. Hence, audio should not be scaled. Secondly, high-quality audio (e.g. multichannel) consumes just a fraction of the resources compared to high-quality video. Hence, scaling audio hardly contributes to cost-effectiveness. The challenge for multimedia QoS for CTs is in finding a QoS approach that can primarily be applied to high-quality video, and also supports other media, such as 3D graphics. The main focus of the V-QoS program has therefore been on QoS for high-quality video.

2.3.2 QoS parameters for high-quality video processing in MCTs

High-quality video processing in MCTs has a number of distinctive characteristics when compared to mainstream multimedia processing in, for example, a (networked) workstation environment [Nahrstedt, Chu & Narayan, 1998]. In this section, we compare QoS parameters for different types of media processing. These QoS parameters are shown to be very application domain specific. Moreover, high-quality video has very stringent timing requirements compared to other media processing functions.

The mesh, texture, and screen resolution are used as QoS parameters for 3D computational graceful degradation by Lafruit et al. [2000], while maintaining a fixed frame rate. Frequently used QoS parameters for video applications in a workstation environment are screen resolution, frame rate (with a maximum of 30 Hz), image size, color depth, bit rate and compression quality [Li & Nahrstedt, 1999; Morros & Marqués, 1999; Sabata, Chatterjee & Sydir, 1998]. Spatial (resolution) and temporal (bit rate and frame rate) scalabilities are exploited in great detail in the field of image compression; see [Morros & Marqués, 1999]. In MCTs with high-quality video requirements, these parameters are not generally applicable. High-quality video has a fixed field/frame-rate of 24 – 60 Hz, no tolerance for jitter (i.e. frame-rate fluctuations), and low tolerance for frame skips, i.e. very stringent timing requirements. It is conceivable, however, that future users will expect guaranteed timing behavior from multimedia applications on desktops and Internet appliances as well [Rajkumar et al., 1998]. Moreover, the resolution of a TV screen is fit to its standard (e.g. PAL, NTSC, ATSC), and the image (or window) size is either fixed (e.g. main window or PiP window) or determined by the user. Finally, receivers in a broadcast environment, and that is what MCTs currently are, do not have the option to negotiate compression quality and bit-rate, although that may change in the future for MCTs in an in-home digital network.

Although the setting of many parameters are imposed by the environment rather than adaptable by the system for optimization purposes, they do determine the amount of processing required for a particular video output quality. As an example, the window size determines the embedded resizing techniques [Zhong & Chen, 2001] that can be applied without loss of visual output quality. Hence, alternative parameters have to be used for high-quality video. These parameters are typically video algorithm specific and may vary per algorithm. Optional parameters for high-quality video are number of filter coefficients (e.g. 0, 8, 32, ...), reference objects (e.g. points, lines, ...), and type of processing (e.g. linear, nonlinear). Hentschel, Braspenning & Gabrani [2001] present a scalable sharpness enhancement algorithm illustrating the exploitation of such parameters. Another example is presented by Peng [2001] and Yanagihara, Sugano, Yoneyama & Nakajima [2000]. They describe scalable MPEG-2 video decoding by computing only a few IDCT coefficients.

2.3.3 Scalable video processing

As illustrated by Figure 1.1 on page 3, an MCT may accept input from different types of input sources, such as satellite, cable, storage devices, Internet and Ethernet. The video input can be digital or analog. An MCT may have a number of video outputs, such as a display, a storage device (such as a video recorder, DVD+RW, or a hard disk), and an IEEE 1394 or Internet link. The outputs on a display may be sub-divided into two (dynamically changing) groups based on user-focus. User focus induces a relative importance on outputs.

Between these inputs and outputs, a number of processing paths may exist, containing joins and forks in complex settings. Each processing path typically consists of a number of functional processing parts, termed *jobs*, e.g. channel decoding, picture enhancement, and rendering (for a display) or encoding (for a link). Jobs inherit the relative importance of the output with the highest relative importance to which they contribute. Whereas the functional description of a job is general (e.g. enhancement), there may be a number of specific algorithms (processing variants) within a single job. One or more jobs constitute an RCE. The single-RCE application *main* of our demonstrator in Figure 2.6 consists of multiple jobs.

The remainder of this section presents aspects of scalable media processing. The first subsection presents the basic structure of a scalable algorithm (SA) for media processing. Scalable video algorithms (SVAs) are a special class of SAs, as exemplified above by their specific scalability parameters. The second subsection considers jobs in more detail. Additional information may be found in [Gabrani, Hentschel, Steffens & Bril, 2003].

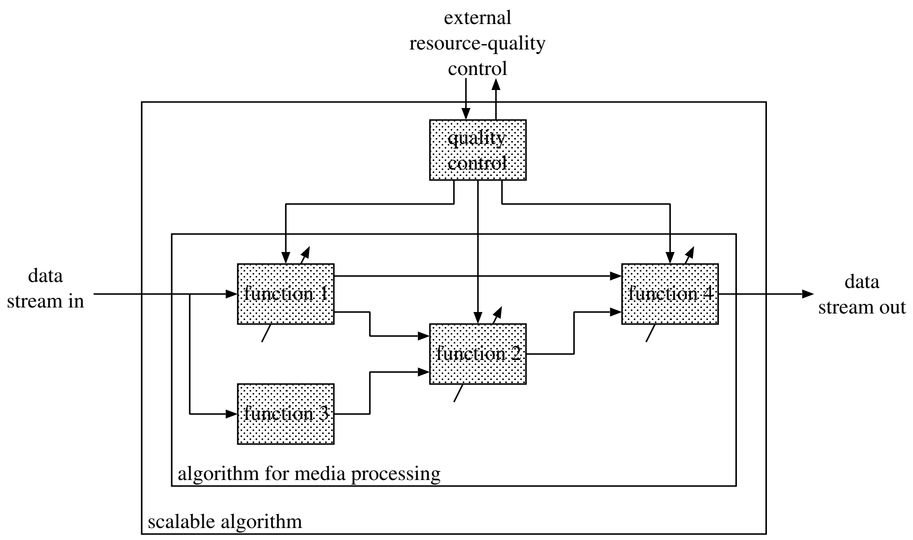


Figure 2.8. Example of the basic structure of a scalable algorithm.

Scalable algorithms

An SA basically consists of an algorithm for media processing and a quality control block [Hentschel, Braspenning & Gabrani, 2001]; see Figure 2.8. The algorithm can be split in a number of specific functions, some of which are scalable. The quality of the output depends on the appropriate combination of the quality levels of these functions.

Of these combinations, only a few provide acceptable quality levels for the SA; see Figure 2.9. The optimal quality-resource combinations, which are termed Pareto-optimal points, are connected by the curve with maximum quality at lowest resources. The quality control block contains this information and the appropriate settings for the functions.

Jobs

A *job* is a flexible set of SVAs. A unique combination of algorithms within a job is termed a *job mode*. The job mode is selected dynamically, and a change of job mode, e.g. due to a channel change or an exchange of the contents of the main window and a PiP window, is termed a *job-mode change* (JMC). Figure 2.10 shows eight examples of job modes for a video enhancement job based on four different formats for the input stream and two different formats for the output stream (main or PiP). A JMC may lead to a change in the specific functionality of the job, and the number and order of its algorithms. When a job is realized in hardware, it typically only has a single job mode, which provides the best approach for all

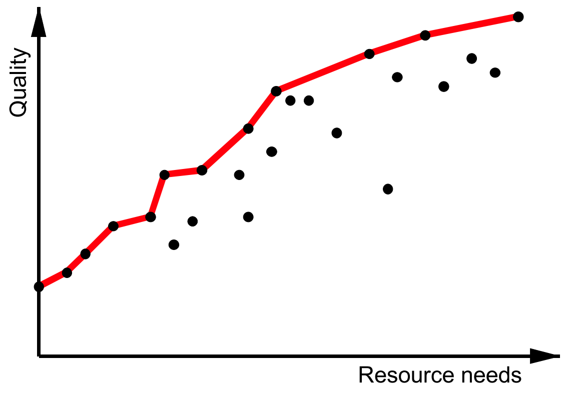


Figure 2.9. Best choices of quality-resource combinations for functions of the entire scalable algorithm.

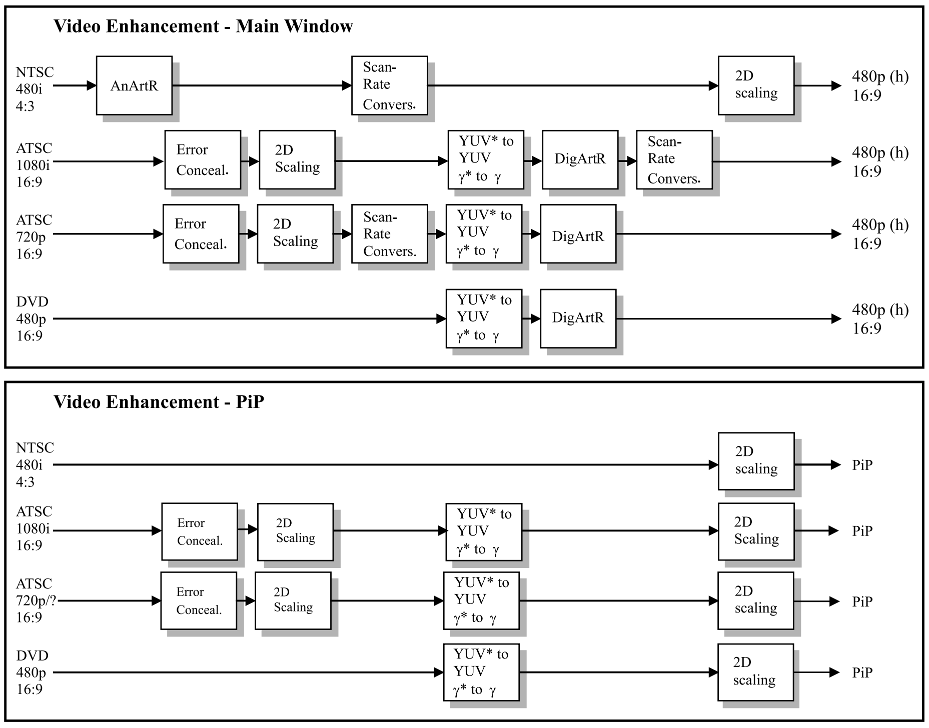


Figure 2.10. Eight modes of a video enhancement job. The job modes depend on the input stream format and the output window (main or PiP).

cases. Media processing in software therefore provides an additional opportunity for quality improvements.

For each job mode, a number of operational sets are defined. Each operational set determines specific processing for each algorithm depending on characteristics such as window size (determining the applicable embedded resizing techniques) and user focus. An operational set is selected dynamically, and a change of operational set is termed *operational-set change*.

Just like a number of functions constitutes an SA (see Figure 2.8), a unique combination of algorithms constitutes a job mode. Similar to SAs, jobs can be scaled, giving rise to a set of discrete quality levels associated with each job mode. Jobs provide (estimated) resource requirements for each quality level. In order to allow system optimization by QoS RM the value of the result of a job is expressed, in semantically neutral terms, as a *job utility*. Each operational set of a job mode consists of a number of quality levels, and an associated *quality mapping*, which determines resource requirements and a job utility for each quality level in the operational set.

2.3.4 QoS control strategies for high-quality video

As mentioned above, applications have to get by with their budget. Frame skipping is a basic degradation technique to control video processing with limited resources that can also be applied to non-scalable video [Hamann, Löser, Reuther, Schönberg, Wolter & Härtig, 2001; Isović, Fohler & Steffens, 2003], and is generally available in MPEG decoders implemented in software. Because high-quality video has a low tolerance for frame skips, frame skipping alone is unacceptable to reduce load upon overloads.

Lan, Chen & Zhong [2001] describe an extension of this basic degradation technique of a scalable MPEG-2 decoder with a method to regulate varying computational load. The method assumes *strictly periodic budgets* and *synchronous processing*, i.e. during each budget period an entire frame must be decoded, and only one frame is decoded per period. Before an MPEG-2 frame is decoded, the required resources are estimated, and the decoding is subsequently scaled such that it will not exceed its resource budget. A similar method has been described by Lafruit et al. [2000] for adaptive control for 3D decoding and rendering.

Although this method prevents frame skips, it only optimizes the quality of individual frames rather than a sequence of frames. Moreover, it does not exploit the opportunity offered by buffering. With buffers, an application may work ahead (or lag behind) compared to its budget consumption, and an MPEG decoder with a strictly periodic budget can therefore process frames *asynchronously*. Wüst & Verhaegh [2001, 2002] present QoS control strategies for high-quality video processing based on asynchronous processing that balance three aspects of user-perceived quality: picture quality, deadline misses, and quality changes. These strategies are able to accommodate both temporal and structural load fluctuations, and allow a close-to-average-case resource allocation to a single video processing task. The interested reader is referred to those papers for further details. Specific control for an RCE consisting of multiple (scalable and non-scalable) tasks, as shown in Figure 2.6, and for an application consisting of multiple RCEs has not been addressed in the literature.

2.4 QoS-based resource management

In this section, we consider QoS-based resource management, being the second pillar of our co-operative QoS approach. This section will be brief, and only describe essential and necessary basics for the chapters to follow. Section 2.4.1 considers the software framework, and Section 2.4.2 illustrates how the software framework ensures seamless switching between modes by means of an example.

2.4.1 Software framework

For ease of presentation, we make the following assumptions. Our basic system supports a set of adaptive applications, where each application consists of a single RCE. The multi-layer control hierarchy consists of just two layers. The lower layer contains the specific controllers of the applications, which reside within the RCEs, and the higher layer contains a single controller, being the so-called 'quality manager' (QM). Moreover, we assume a basic reservation-based resource manager, that contains a so-called 'budget scheduler' (BS).

The QM selects quality levels at which the RCEs are executed and allocates CPU budgets to RCEs in such a way that the overall (system) *utility* is maximized, and the estimated resource requirements meet the resource availability. Next to performing this global optimization, the QM maintains the quality mappings of the running RCEs based on the actual resource needs measured by the BS. Changes in the number of applications, relative importance of the applications, and quality mappings of the RCEs require re-optimizations.

The BS provides CPU budgets to RCEs. These budgets are time-triggered and strictly periodic. Budgets are implemented by means of priority manipulations. Budget accounting and enforcement is based on timers. In Chapter 7, we describe how the BS can be extended with CGBs. A concise description of the implementation of the BS is given by Bril & Steffens [2001], and will be provided in Section 2.5.

The existing BS is based on RMS, therefore suffers from *scheduling imperfections*, and the system will typically have slack time. Moreover, it is conceivable to explicitly reserve (slack) time for overload handling through judicious spare-

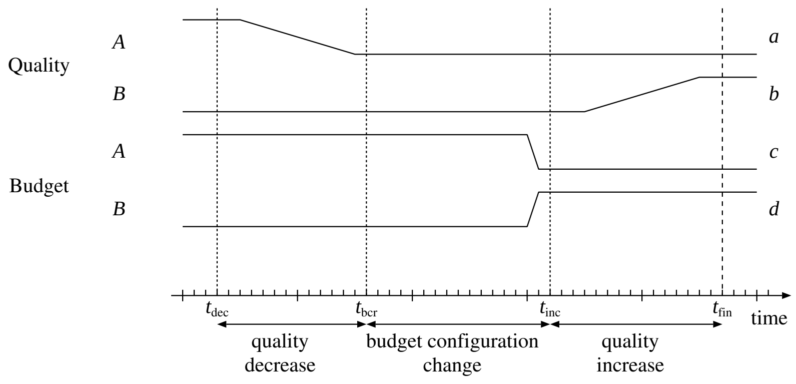


Figure 2.11. An example of a mode change. At time t_{dec}, the system instructs application A to decrease its operational quality level. The decrease results in a smooth degradation of the quality of A's output (line a), and A informs the system that it completed the quality change at time t_{bcr}. The system subsequently reallocates the resources by means of a budget configuration change (lines c and d). At time t_{inc}, the BCC has been completed, and the system instructs application B to increase its operational quality level. The increase results in a smooth improvement of the quality of B's output (line b). At time t_{fin}, B has completed the quality change, and it may inform the system about this, which completes the mode change.

capacity allocation. Although spare-capacity allocation is an essential part of any approach that aims at optimizing systems with scarce resources, the existing implementation has no policy for spare-capacity allocation, only supports default spare-capacity provisioning, and does not account for spare-capacity execution. In Chapter 7, we sketch how our design and implementation of CGBs can be complemented with mechanisms for spare-capacity provisioning.

2.4.2 Ensuring seamless switching between modes

Consider a system with two applications, A and B, that both have results that are immediately visible (i.e. end results). Assume a change of user focus from A to B, requiring a re-optimization of the system [Ott et al., 1998]. For ease of presentation, we assume that this change involves no application mode change for either of the applications. When the system is notified about this change of user focus, it first determines the new optimal quality levels at which the applications have to run, and subsequently performs a mode change. The mode change is shown in Figure 2.11. Note that by allowing A to smoothly degrade its quality level, this mode change is in the spirit of the mode change protocol described by Tindell, Burns & Wellings [1992]. The budget configuration change (BCC) shown in Figure 2.11 is based on [Tindell & Alonso, 1996]. The system waits for an *idle interval*, i.e. a time interval in which no RCE can execute based on its own budget, which ensures that the BCC will not affect the guarantees of budgets.

2.5 Budget scheduler

This section describes the basic implementation of the budget scheduler (BS) and the motivation for on-line calculations of an admission test. We start with a brief description of a basic model for budgets.

2.5.1 A basic model for budgets

Budgets are periodic, and the budget period may be different for each RCE. The budget for RCE ρ_i is denoted by a triple (B_i, T_i, φ_i), where T_i is the budget period, φ_i is the phasing, and B_i the budget per period for ρ_i.

2.5.2 Priority bands

In-budget execution is performed at high priority, and out-of-budget execution, i.e. an execution based on spare capacity, is done at low priority. This gives rise to two main priority bands, a high-priority band (HP) for in-budget executions and a low-priority band (LP) for out-of-budget executions. An RCE that consists of multiple tasks gives rise to a sub-priority band, so that tasks within the RCE can be prioritized. Priority bands of RCEs are disjoint (i.e. they do not overlap). This is illustrated in Figure 2.12, where RCE ρ_1 consists of four tasks, hence ρ_1 gives rise to a sub-priority band containing four priority levels, and RCE ρ_2 consists of a single task.

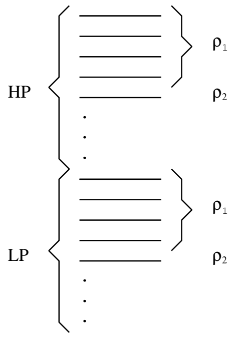


Figure 2.12. Priority bands for implementation of budgets, a high-priority band (HP) for in-budget executions and a low priority band (LP) for out-of-budget executions.

2.5.3 Priority manipulation

In both HP and LP, the RCEs are scheduled in rate-monotonic priority order, i.e. RCEs with smaller budget periods get higher priorities. At the start of each new period, the budget is replenished, and the priority of an RCE is raised to its rate-monotonic priority within HP. When the budget is exhausted, the RCE's priority is lowered to LP. In case of a multi-task RCE, the complete sub-priority band is raised or lowered, leaving the internal priority ordering intact.

Without appropriate precautionary measures, self-suspension of an RCE may result in deferred execution, which may jeopardize budget guarantees. The existing BS therefore withdraws the AGB of an RCE when the RCE releases the processor, i.e. the priority of the RCE is lowered to LP. The remainder of the budget becomes available as spare capacity, i.e. for out-of-budget execution.

Whereas a synchronous RCE releases the processor when it completes its work, a streaming (i.e. asynchronous) RCE never completes. Instead, a streaming RCE tries to work ahead as much as possible till it blocks on either input or output. When either new data arrives at its input or space becomes available in its output, the RCE becomes un-blocked again and resumes its execution. As a consequence, this behavior may also result in a deferred execution. A streaming RCE typically only blocks when it is ahead, and the existing BS therefore withdraws the AGB of a streaming RCE upon blocking. An additional advantage of this implementation is that frequent context switches due to blocking and un-blocking of a streaming RCE, giving rise to unnecessary overhead, are prevented.

2.5.4 Admission tests

In this section, we briefly describe the motivation for on-line calculations of response times and occupied times as part of the evaluation of an admission test. Remember that the CPU resource reservation of the budget scheduler is based on fixed-priority scheduling (FPS) and the admission test is based on an *exact* schedulability test as derived by Joseph & Pandya [1986] and Audsley, Burns, Richardson & Wellings [1991]. We start with the need for an online evaluation of a schedulability test. We subsequently consider the need for efficient schedulability tests. Finally,we describe the need for response time and occupied time calculations to perform a schedulability test.

Need for online evaluation

Many real-time systems become increasingly dynamic, and may run in many different modes that are not predictable or known statically. As a consequence, these systems demand for an on-line evaluation of schedulability tests. A special class of such systems is given by high-volume electronics (HVE) consumer terminals, such as digital TV sets, digitally improved analogue TV sets and set-top boxes

(STBs), performing media processing in software using dedicated media processors. This class of systems and a Quality-of-Service approach for next generations of HVE consumer products aiming at cost-effective media processing in software has been described in this chapter. In that approach, multiple evaluations of the schedulability test are typically required to maximize perceived quality.

Complexity versus effectiveness

Unfortunately, *exact* schedulability tests are complex, i.e. not efficient. Although the complexity of a schedulability test is not a major issue when the test can be evaluated statically, i.e. off-line, it may be prohibitive when it needs to be evaluated on-line. Therefore, Bini et al. [2001] recommend the usage of pessimistic tests, that have an $O(n)$ complexity, when exact tests cannot be applied for efficiency reasons. Many real-time systems performing an on-line schedulability test require exact tests, and using pessimistic tests is therefore not appropriate. For cost-effectiveness reasons, we strive for a processor utilization for next generations of HVE consumer terminals that exceeds the utilization bounds mentioned by Bini et al. [2001], and therefore aim at reducing the cost of exact tests.

Need for response time and occupied time calculations

Scheduling algorithms and schedulability tests are indissolubly connected. Obviously, a schedulability test also depends on (the sophistication of) the real-time scheduling model, e.g. whether or not blocking and jitter are taken into account or arbitrary deadlines are covered. Moreover, the scheduling model influences whether or not response times and occupied times have to be determined in order to perform an *exact* test. This will be illustrated by a number of examples.

There exist exact tests for FPS under arbitrary phasing without the need to calculate worst-case response times [Lehoczky, Sha & Ding, 1989; Bini & Buttazzo, 2002]. With arbitrary deadlines, we however do need to determine the worst-case response times, i.e. the worst-case response times of all activations in a busy period [Lehoczky, 1990]. When a model includes jitter, we also need best-case response times, as illustrated in Chapter 9. Finally, in order to analyze a system with CGBs, we need occupied times next to response times; see Chapter 8.

Because of this need for response times and occupied times to perform a schedulability test, we consider the cost of calculating those values in Chapter 6.

3

Real-Time Scheduling Basics

This chapter presents the basic terminology and concepts of real-time scheduling of a set of independent periodic tasks on a single processor. The focus will be on *preemptive* scheduling, i.e. a form of scheduling where the execution of tasks can be interrupted at any time. The order in which the tasks are executed by the processor is determined by a *scheduling algorithm*. The specific operation of allocating the processor to a task selected by the scheduling algorithm is termed *dispatching*. We assume dispatching to be based on priorities of tasks, and priority assignment to tasks according to a scheduling algorithm. A scheduling algorithm is classified to be *static* (or *fixed*), when the priority assignment is fixed before the execution of the tasks. Otherwise the algorithm is classified to be *dynamic*.

This chapter has the following structure. We start with a real-time scheduling model in Section 3.1. Most of the definitions and assumptions of this model originate from [Liu & Layland, 1973]. The *processor utilization factor*, which also originates from [Liu & Layland, 1973], is introduced in Section 3.2. Examples of static and dynamic priority scheduling algorithms are subsequently briefly described in Section 3.3. In Section 3.4, we revisit the assumptions of the scheduling model.

3.1 A real-time scheduling model for tasks

We assume a single processor and a set $\mathcal{T}$ of n independent periodic tasks, denoted by $\tau_1, \tau_2, \ldots \tau_n$. At any moment in time, the processor is used to execute the highest priority task that has work pending. So, when a task τ_i is being executed, and a release occurs for a higher priority task τ_j, then the execution of τ_i is preempted, and will resume when the execution of τ_j has ended, as well as all other releases of tasks with a higher priority than τ_i that have taken place in the meanwhile.

A *schedule* is an assignment of the tasks to the processor. A schedule can be defined as an integer step function $\sigma : \mathbb{R} \to \{0, 1, \ldots, n\}$ [Buttazzo, 2002]. Informally, $\sigma(t) = k$ with $k > 0$ means that task τ_k is being executed at time t, while $\sigma(t) = 0$ means that the processor is idle. More formally, σ partitions the timeline in a set of non-empty, right semi-open intervals $\{[t_j, t_{j+1})\}_{j \in \mathbb{Z}}$, such that $\sigma(t)$ is right-continuous and piece-wise continuous in each of those intervals, and discontinuous at the ends. At times t_j, the processor performs a *context switch*. Figure 3.1 shows an example of the execution of a set $\mathcal{T}$ of three periodic tasks and the corresponding value of the schedule $\sigma(t)$. The *level-i schedule* σ_i is an assignment of the tasks τ_1 till τ_i to the processor.

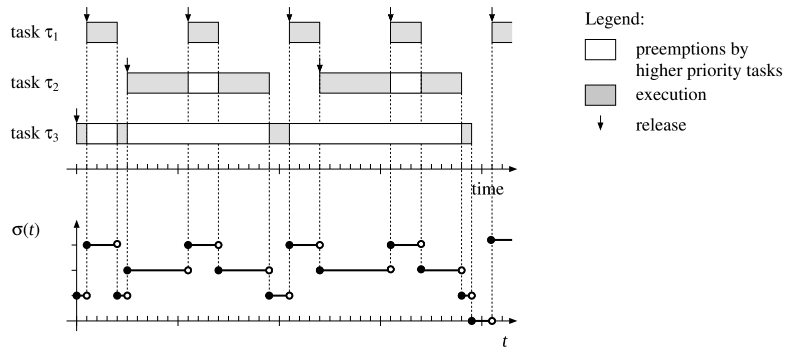


Figure 3.1. An example of the execution of a set $\mathcal{T}$ of three independent periodic tasks τ_1, τ_2, and τ_3, where task τ_1 has highest priority, and task τ_3 has lowest priority, and the corresponding value of $\sigma(t)$.

The remainder of this section is organized as follows. We start by describing task characteristics in Section 3.1.1. In Section 3.1.2, we introduce scheduling related terms denoting specific moments in time and intervals of time. We stick to the terminology used by Buttazzo [2002]. Dual notions for worst-case and best-case situations are presented in Sections 3.1.3 and 3.1.4, respectively. Finally, in Section 3.1.5, we describe the assumptions we make on the environment.

3.1.1 Task characteristics

Each task τ_i is characterized by a (*release*) *period* $T_i \in \mathbb{R}^+$, a *worst-case computation time* $WC_i \in \mathbb{R}^+$, a *best-case computation time* $BC_i \in \mathbb{R}^+$, and a (*relative*) *deadline* $D_i \in \mathbb{R}^+$, where $BC_i \leq WC_i \leq min(D_i, T_i)$. The *activation* (*release* or *request*) *time* is the time at which a task τ_i becomes ready for execution. An activation of a task is also termed a *job*. The job of task τ_i with activation time $\varphi_i \in \mathbb{R}$, serves as reference activation. Depending on the context, this job is referred to as either job zero or job one. When referred to as job zero, the activation of job k of τ_i takes place at time $a_{ik} = \varphi_i + kT_i$, $k \in \mathbb{Z}$. The activation time φ_i is also termed the *phasing* of task τ_i. The combination of phasings φ_i is termed the phasing φ of the task set $\mathcal{T}$. We assume that we do not have control over the phasing φ, for instance since the tasks are released by external events, so we assume that any arbitrary phasing may occur.

3.1.2 Times and intervals

The (*absolute*) *deadline* of job k of τ_i takes place at time $d_{ik} = a_{ik} + D_i$. The *begin* (or *start*) *time* b_{ik} and *finalization* (or *completion*) *time* f_{ik} of job k of τ_i is the time at which τ_i actually starts and ends the execution of that job, respectively.

The *start interval* of job k of τ_i is defined as the time span between the activation time of the job and its begin, i.e. $[a_{ik}, b_{ik})$. The length of the start interval is denoted by s_{ik}, hence $s_{ik} = b_{ik} - a_{ik}$. The *response interval* of job k of τ_i is defined as the time span between the activation time of that job and its completion, i.e. $[a_{ik}, f_{ik})$. The length of the response interval is denoted by r_{ik}, hence $r_{ik} = f_{ik} - a_{ik}$. The *execution interval* of job k of τ_i is defined as the time span between the begin time of the job and its completion, i.e. $[b_{ik}, f_{ik})$. The length of the execution interval is denoted by e_{ik}, hence $e_{ik} = f_{ik} - b_{ik}$. Note that $r_{ik} = s_{ik} + e_{ik}$.

Figure 3.2 illustrates the above basic notions for an example job k of task τ_i. In this figure, the job is preempted by higher priority tasks in the intervals corresponding to the white boxes, i.e. $[a_{ik}, b_{ik})$ and $[b_{ik} + x_1, f_{ik} - x_2)$. The job executes without interruption in the disjoint intervals corresponding to the grey boxes, i.e. $[b_{ik}, b_{ik} + x_1)$ and $[f_{ik} - x_2, f_{ik})$. The actual execution time of the job, $x_1 + x_2$, satisfies $BC_i \leq x_1 + x_2 \leq WC_i$. Time $b_{ik} + x_1$ is termed a *preemption time* of job k of task τ_i and time $f_{ik} - x_2$ is termed a *resume time* of job k of τ_i.

As mentioned above, a schedule σ partitions the timeline in a set of non-empty, right semi-open intervals $\{[t_j, t_{j+1})\}_{j \in \mathbb{Z}}$. An *idle interval* is defined as a maximal interval during which the processor is idle, i.e. $[t_j, t_{j+1})$ is an idle interval when $\sigma(t) = 0$ for $t \in [t_j, t_{j+1})$, $\lim_{t \uparrow t_j} \sigma(t) \neq 0$, and $\sigma(t_{j+1}) \neq 0$. A *level-i idle interval* is defined as a maximal interval during which the processor is idle in a level-i schedule σ_i, i.e. $[t_j, t_{j+1})$ is a level-i idle interval when $\sigma_i(t) = 0$ for $t \in [t_j, t_{j+1})$, $\lim_{t \uparrow t_j} \sigma_i(t) \neq 0$, and $\sigma_i(t_{j+1}) \neq 0$. Similarly, a *busy interval* and a *level-i busy*

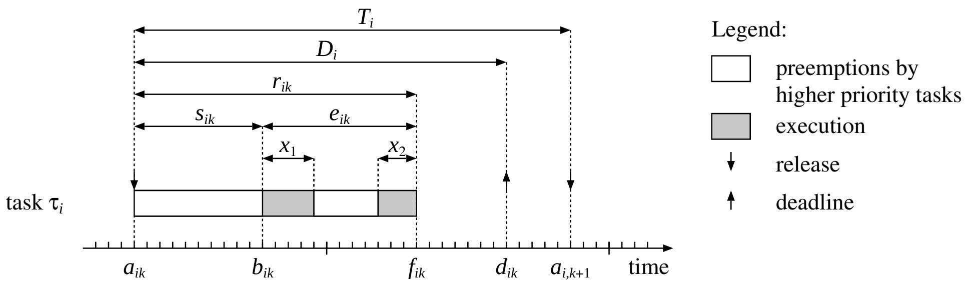


Figure 3.2. Basic model for task τ_i.

interval are defined as a maximal interval during which the processor is non-idle in a schedule σ and a level-i schedule σ_i, respectively. From a processor perspective, the timeline may be viewed to be constituted from alternating idle and busy intervals.

3.1.3 Worst-case notions

Next, we present worst-case notions. A *critical instant* of a task is defined to be an (hypothetical) instant that leads to the largest response interval for that task. The largest response interval is termed the *worst-case response interval.* The length of the worst-case response interval of task τ_i is denoted by WR_i, i.e.

$$WR_i = \sup_{\varphi,k} r_{ik}. \tag{3.1}$$

It is common practice to use the term *worst-case response time* as a synonym for the length of the worst-case response interval.

Similarly to the largest response interval, we define notions for the largest start interval and the largest execution interval. These latter intervals are termed the *worst-case start interval* and the *worst-case execution interval*, and their lengths for task τ_i are denoted by WS_i and WE_i, respectively, i.e.

$$WS_i = \sup_{\varphi,k} s_{ik}, \tag{3.2}$$

$$WE_i = \sup_{\varphi,k} e_{ik}. \tag{3.3}$$

We will use use the terms *worst-case start time* and *worst-case execution time* as a synonyms for the length of the worst-case start interval and the length of the worst-case execution interval, respectively.

3.1.4 Best-case notions

The best-case notions are duals of the worst-case notions. An *optimal instant* of a task is defined to be an (hypothetical) instant that leads to the shortest response interval. The shortest response interval is termed the best-case response interval. The length of the best-case response interval of task τ_i is denoted by BR_i, i.e.

$$BR_i = \inf_{\varphi,k} r_{ik}. \tag{3.4}$$

We will use the term *best-case response time* as a synonym for the length of the best-case response interval.

Similarly, to the shortest response interval, we define notions for the shortest start interval and the shortest execution interval. These latter intervals are termed the *best-case start interval* and the *best-case execution interval*, and their lengths for task τ_i are denoted by BS_i and BE_i, respectively, i.e.

$$BS_i = \inf_{\varphi,k} s_{ik}, \tag{3.5}$$

$$BE_i = \inf_{\varphi,k} e_{ik}. \tag{3.6}$$

We will use use the terms *best-case start time* and *best-case execution time* as a synonyms for the length of the best-case start interval and the length of the best-case execution interval, respectively.

3.1.5 Basic assumptions

The following basic assumptions are made on the environment. Most of these basic assumptions originate from Liu & Layland [1973] and are common in the literature.

1. Tasks have unique priorities.
2. Tasks will be preempted instantaneously when a higher priority task becomes ready to run.
3. Tasks are ready to run at the start of each period and do not suspend themselves.
4. Tasks are independent, i.e. there is no task synchronization.
5. The overhead of context switching and task scheduling is ignored.
6. Job $k+1$ of task τ_i does not start before the end of job k, i.e. $f_{ik} \leq b_{i,k+1}$.
7. No specific phasing of tasks at start-up is assumed.

Finally, we assume that the deadlines are hard, i.e. each job of a task must be completed before its deadline. Hence, a set $\mathcal{T}$ of n periodic tasks can be scheduled if and only if

$$WR_i \leq D_i \tag{3.7}$$

for all $i = 1, \ldots, n$.

In most examples we use, the worst-case and best-case computation times are chosen to be equal ($WC_i = BC_i$), and we simply use the term *computation time* C_i.

To prevent verbosity, we typically use the phrase 'the interval x' rather than 'the length x of the interval', and 'the interval' rather than 'the length of the interval', e.g. 'the worst-case response interval WR_i' rather than 'the length WR_i of the worst-case response interval'. In many cases, we are not interested in the value of an interval of a task for a particular computation time, but in the value as a function of the computation time. We will therefore use a functional notation for intervals when needed, e.g. $WR_i(C_i)$.

3.2 Utilization factors

Given a set $\mathcal{T}$ of n periodic tasks, the (*processor*) *utilization factor* U is the fraction of the processor time spent on the execution of the task set [Liu & Layland, 1973]. The fraction of processor time spent on executing task τ_i is C_i/T_i, and is termed the *utilization factor* U_i^τ of task τ_i, i.e.

$$U_i^\tau = \frac{C_i}{T_i}. \tag{3.8}$$

The *cumulative utilization factor* U_i for tasks τ_1 till τ_i is the fraction of processor time spent on executing these tasks, and is given by

$$U_i = \sum_{j \le i} U_j^\tau. \tag{3.9}$$

Therefore, U is equal to the cumulative utilization factor U_n for n tasks.

$$U = U_n = \sum_{j \le n} U_j^\tau = \sum_{j \le n} \frac{C_j}{T_j}. \tag{3.10}$$

When the best-case and worst-case computation times are different, we have best-case and worst-case utilization factors in addition, i.e. the *best-case utilization factor* $BU_i^\tau = BC_i/T_i$ and the *worst-case utilization factor* $WU_i^\tau = WC_i/T_i$ of task τ_i, the *best-case cumulative utilization* $BU_i = \sum_{j \le i} BU_j^\tau$ and the *worst-case cumulative utilization* $WU_i = \sum_{j \le i} WU_j^\tau$ of tasks τ_1 till τ_i, and the *best-case* (*processor*) *utilization factor* $BU = BU_n$ and the *worst-case* (*processor*) *utilization factor* $WU = WU_n$ of the task set $\mathcal{T}$. Note that the cumulative utilization of an empty task set is zero, i.e. $BU_0 = U_0 = WU_0 = 0$.

3.3 Scheduling algorithms

Liu and Layland determined the following necessary condition for the schedulability of a set $\mathcal{T}$ of n periodic tasks under any scheduling algorithm.

$$U \le 1. \tag{3.11}$$

Unless explicitly stated otherwise, we assume in this thesis that task sets satisfy this condition. Whether or not the condition is also sufficient depends on the particular scheduling algorithm. Scheduling algorithms and schedulability tests for set of tasks are indissolubly connected, i.e. every scheduling algorithm has a dedicated schedulability test. Below, we will briefly describe examples of static and dynamic scheduling algorithms with their accompanying schedulability tests.

3.3.1 Fixed-priority preemptive scheduling

We describe two basic algorithms for fixed-priority preemptive scheduling (FPPS) in this section, being *rate monotonic* (RM) *scheduling* and *deadline monotonic* (DM) *scheduling*. Moreover, we present three main classes of schedulability tests for FPPS.

For notational convenience, we assume for FPPS that in the set $\mathcal{T}$ of n tasks under consideration, task τ_j has a higher priority than task τ_i if and only if $j < i$.

Rate monotonic scheduling

For the RM scheduling algorithm, it is assumed that the deadlines D of all the n tasks of the set $\mathcal{T}$ equal their periods, i.e. $D_i = T_i$ for $i = 1, \ldots, n$. According to the RM scheduling algorithm, each task is assigned a priority inversely proportional to its period, i.e. the tasks are sorted such that $T_i \leq T_{i+1}$ for $1 \leq i < n$. Thus, at any instant, the runnable task with the smallest period is executed. Since periods are constant, application of the RM scheduling algorithm results in a static priority assignment.

Figure 3.3 shows an example of a schedule with an RM priority assignment to tasks. Characteristics of the example are given in Table 3.1. Because the 'task set' of that table satisfies (3.7) under RM priority assignment of tasks, it can be scheduled under RM scheduling.

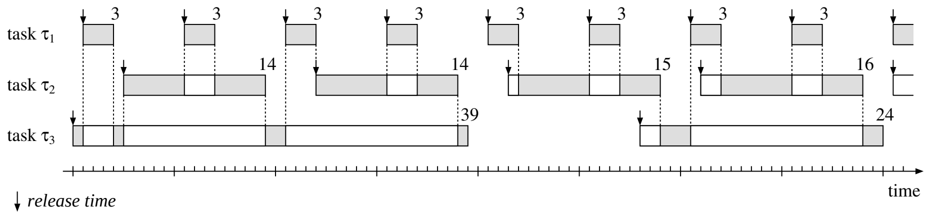


Figure 3.3. An example of the effect of RM scheduling on the execution of three periodic tasks τ_1, τ_2, and τ_3, where task τ_1 has highest priority, and task τ_3 has lowest priority. The numbers to the top right corner of the boxes denote the response times of the respective releases. Note that response time is counted from the moment of release up to the corresponding completion.

Table 3.1. Characteristics of the example of Figure 3.3, that are also used to illustrate the calculation of worst-case response times in Figure 4.4 and best-case response times in Figure 5.4. In this example, best-case computation times are equal to worst-case computation times, and deadlines are equal to periods. Moreover, arbitrary phasings are assumed. Worst-case response times and best-case response times under RM priority assignment are included.

task	period	computation time	utilization factor	worst-case response time	best-case response time
τ_1	10	3	0.3	3	3
τ_2	19	11	0.58	17	14
τ_3	56	5	0.09	56	22

[Liu & Layland, 1973] showed that RM scheduling is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM, under the above assumption of $D_i = T_i$. They also determined the following sufficient (but not necessary) condition for the schedulability of a set $\mathcal{T}$ of n periodic tasks under RM scheduling.

$$U_n \leq n(2^{1/n} - 1) \tag{3.12}$$

We will refer to this test as LL(n). The term $n(2^{1/n} - 1)$ in (3.12) is strictly decreasing as a function of n from 0.83 when $n = 2$, to $\ln 2 \approx 0.693$ as $n \to \infty$. For tasks τ_1 and τ_2 of our example, LL(2) fails, as $U_2 = 0.88 > 0.83$. Because the term $n(2^{1/n} - 1)$ in LL(n) is strictly decreasing, LL(3) fails irrespective of the characteristics of τ_3.

Recently, a so-called hyperbolic bound HB(n) has been presented by Bini et al. [2001] for the RM algorithm.

$$\prod_{j \leq n} (U_j^\tau + 1) \leq 2 \tag{3.13}$$

HB(n) is less pessimistic than LL(n), but is also only a sufficient and not a necessary schedulability test. For tasks τ_1 and τ_2 of our example, HB(2) fails, as $\prod_{j\leq 2}(U_j^\tau + 1) = 2.054 > 2$. Similarly to LL, HB(3) also fails, because the term $\prod_{j\leq n}(U_j^\tau + 1)$ in HB(n) is strictly increasing when tasks are added.

Deadline monotonic scheduling

Leung & Whitehead [1982] proposed the DM scheduling algorithm as an extension to RM scheduling. Whereas RM scheduling assumes that the deadlines of tasks equal their periods ($D_i = T_i$), the DM scheduling algorithm assumes that the deadlines of tasks are smaller than or equal to their periods ($D_i \leq T_i$). According to the DM scheduling algorithm, each task is assigned a priority inversely proportional to its deadline. Thus, at any instant, the runnable task with the shortest (relative) deadline is executed. Since deadlines are constant, application of the DM scheduling algorithm also results in a static priority assignment. Note that the DM scheduling algorithm specializes to the RM scheduling algorithm when the deadlines of tasks equal their periods.

Leung & Whitehead also showed that DM scheduling is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithm can schedule a task set with deadlines smaller than or equal to their periods that cannot be scheduled by DM.

The following variant of (3.12) can be used as a sufficient (but not necessary) condition for the schedulability of a set $\mathcal{T}$ of n periodic tasks under DM scheduling.

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1) \tag{3.14}$$

Schedulability tests

For FPPS, there are three main classes of schedulability tests:

- *optimistic* (i.e. necessary but insufficient) tests, such as the test based on the processor utilization factor, e.g. (3.11);
- *pessimistic* (i.e. sufficient but not necessary) tests, such as those based on *pessimistic bounds*, e.g. (3.12), (3.13), and (3.14); and
- *exact* (i.e. both sufficient and necessary) tests, such as those presented by Audsley et al. [1991] and Joseph and Pandya [1986] that explicitly check (3.7).

In this thesis, we will call these exact tests response-time analysis (RTA) tests. The schedulability condition given by (3.7) is based on the worst-case response times of the tasks of $\mathcal{T}$, and can be used for any fixed priority assignment. There also exist RTA tests for RM and DM scheduling under arbitrary phasing that do not calculate worst-case response times; see [Lehoczky et al., 1989; Bini and Buttazzo, 2002].

Lehoczky et al. [1989] showed that it is not uncommon for large task sets with a processor utilization factor around 0.90 to be schedulable with the RM algorithm. We showed for our example that both pessimistic tests $\mathrm{LL}(n)$ and $\mathrm{HB}(n)$ for the RM algorithm failed for only two tasks τ_1 and τ_2 with a processor utilization factor $U_2 = 0.88$. Hence, although both pessimistic tests are very efficient, i.e. have an $O(n)$ complexity, they are considerably less effective as RTA tests. Conversely, RTA tests are effective, but are considerably less efficient than pessimistic tests. We will return to the efficiency of RTA tests in Chapter 6.

3.3.2 Dynamic priority scheduling

Liu & Layland [1973] also describe a dynamic scheduling algorithm which they call *deadline driven scheduling algorithm*. According to that algorithm, each task is assigned a priority inversely proportional to the deadline of its current request. Thus, at any instant, the task with the earliest absolute deadline has the highest priority, and therefore executes. The algorithm is also termed *earliest deadline first* (EDF) *scheduling algorithm*.

Liu & Layland [1973] also showed that EDF is optimal in the sense of schedulability, i.e. if there exists a feasible schedule for a set of periodic tasks, then EDF can schedule that set. In particular, a set $\mathcal{T}$ of periodic tasks is schedulable by EDF if and only if $\mathcal{T}$ satisfies (3.11).

3.4 Discussion

Below, we revisit the assumptions of the scheduling model as presented in Section 3.1, and briefly discuss variants of and extensions to the scheduling model. We conclude this section with a note on rate-monotonic analysis.

3.4.1 Assumptions revisited

As mentioned in Section 3.3, we assume that task sets satisfy the necessary scheduling condition given by (3.11), i.e. $U \leq 1$. We further assume that deadlines of tasks are smaller than or equal to their periods

$$D_i \leq T_i. \tag{3.15}$$

Cases where deadlines exceed periods fall outside the scope of this thesis. The interested reader is referred to [Lehoczky, 1990], one of the first articles in which the problem of FPPS of periodic tasks with arbitrary deadlines was considered.

As mentioned in Section 3.3.1, we assume for FPPS that in the set $\mathcal{T}$ of n tasks under consideration, task τ_j has a higher priority than task τ_i if and only if $j < i$.

3.4.2 Model variants

In this chapter, we made a number of assumptions about our model. In particular, we assumed preemptive scheduling, and we introduced a model in which all task parameters are reals. In Chapters 6 and 9 we will use two variants of our model. These variants are briefly described below.

Preemptive and non-preemptive scheduling

In this thesis, we are primarily interested in preemptive schedules, i.e. schedules in which a context switch may occur at an arbitrary moment during the execution of a task, and instantaneous preemption is therefore one of the basic assumptions given in Section 3.1.5. In Chapter 9, we also consider non-preemptive schedules, and schedules with deferred preemptions. For these kinds of schedules, a context switch may only occur at specific moments during the execution of a task.

Continuous and discrete scheduling

In Section 3.1, we introduced a model in which all task parameters are reals, and preemptions are allowed at any time. Alternatively, we may consider a model in which all task parameters are integers, i.e. $T_i, WC_i, BC_i, D_i \in \mathbb{Z}^+$ and $\varphi_i \in \mathbb{Z}$ for $1 \leq i \leq n$, and preemptions are restricted to integer time points. Conform the terminology used by Baruah, Rosier & Howell [1990], we will term scheduling based on the latter model *discrete scheduling*, and term scheduling based on models that do not exclusively use integer values *continuous scheduling*. Baruah et al. argue in favor of a discrete model. After ample discussion, they conclude that integer task parameters are the proper abstraction of the underlying physical problem, and that restricting preemptions to integer values is preferable. The reason for introducing and using the non-discrete model is that it eases the proofs of theorems and lemmas, and that the results can equally well be applied for a discrete model.

In Chapter 6, where we consider the efficiency of calculating response times, amongst others, we base our analysis on discrete scheduling.

3.4.3 Model extensions

In this chapter, we assumed that tasks are released *strictly* periodically. In Chapter 9, we also consider *jitter* (i.e. variance from the periodicity). Rather than considering periodic tasks only, we can also extend our model with *sporadic* tasks [Mok, 1983]. A sporadic task is characterized by a *minimal* interarrival time, i.e. by a *worst-case* (release) period WT rather than a fixed (release) period T. As discussed by Audsley, Burns, Richardson, Tindell & Wellings [1993], worst-case analysis does not change by extending the basic real-time scheduling model with sporadic tasks. Liu [2000] therefore defines a periodic task as a task with a (bounded) minimal interarrival time. Though sufficient for worst-case analysis, we need to revisit sporadic tasks for best-case analysis. In particular, we need the dual notion of a *maximal* interarrival time for best-case analysis, i.e. a *best-case* (release) period BT, where $WT \leq BT$. Given this notion, we can subsequently use a similar argument as Audsley, Burns, Richardson, Tindell & Wellings [1993] to show that best-case analysis does not change by extending the basic real-time scheduling model with sporadic tasks. For ease of presentation, we exclude sporadic tasks from our model, however.

We also assumed that tasks are independent, i.e. do not interact. This restriction from [Liu & Layland, 1973] has been removed by the priority ceiling protocol [Sha et al., 1990] (and other similar protocols such as the stack resource protocol [Baker, 1991]). The interested reader is referred to [Audsley, Burns, Richardson, Tindell

& Wellings, 1993], showing how blocking of a task by lower priority tasks can be handled. We will use these results in Chapter 9 when we consider non-preemptive schedules and schedules with deferred preemptions.

3.4.4 A note on rate-monotonic analysis

Rate-monotonic analysis (RMA) provides a set of analytical and synthetic methods for building systems that meet hard real-time requirements (deadlines), using fixed-priority preemptive scheduling (FPPS). FPPS is the de-facto standard for real-time scheduling, and is supported by commercially available real-time operating systems. RMA has been adopted by leading companies and institutions world-wide [Obenza, 1994]. It has been used in the entire range from simple control applications to large defense and aero-space applications. Carnegie Mellon University's Software Engineering Institute (SEI) has produced a Practitioner's Handbook for Real-Time Analysis [Klein et al., 1993], based on RMA, which is largely devoted to the problem of casting a real-time system under investigation into the framework of the theoretical model.

4

Worst-Case Analysis

This chapter presents theorems for the main worst-case notions introduced in Chapter 3, i.e. *critical instant* in Section 4.1, *worst-case response time* in Section 4.2, and *worst-case start time* and its generalization *worst-case occupied time* in Section 4.3. Whereas the theorems for the notions of critical instant and worst-case response time are known, the theorems for the other notions are novel. Moreover, the proofs for the existing theorems are novel. The main reason for providing these existing theorems together with their novel proofs is to show the duality of the worst-case notions and the best-case notions presented in the next chapter.

4.1 A critical instant

The next theorem introduces the notion of a *critical instant*, as depicted in Figure 4.1. The theorem states that if we want to determine the worst-case response time under arbitrary phasings, it suffices to consider critical instants only. This notion is used in the next section to determine the worst-case response time. The worst-case response time analysis is based on worst-case computation times.

First, we prove the following two lemmas.

Lemma 4.1. *If an execution of task τ_i, which is released (i.e. activated) at time a, has maximal response time, then the following two properties hold.*

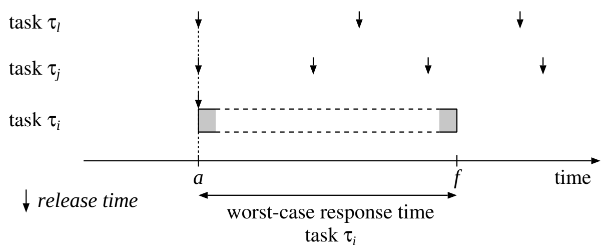


Figure 4.1. A critical instant for task τ_i, at time a, where releases for all higher priority tasks (here τ_j and τ_l) coincide with the release of an execution of τ_i.

(i) No higher priority task has a completion (i.e. finalization) at time a.

(ii) The release of the considered execution of τ_i is immediately preceded by a non-empty sub interval of a level-($i-1$) idle interval.

Proof. (i) Let the release of the considered execution at time a coincide with the completion of a higher priority task which is released at time a'. Then we know that the processor is busy executing that higher priority task or another task with a higher priority than τ_i in the interval $[a', a)$. Hence, we can increase the response time of the considered execution by an amount $a - a'$ by shifting its release to the left by an amount $a - a'$, i.e. by changing the phasing φ into a phasing φ' with $\varphi'_i = \varphi_i + a' - a$ and $\varphi'_j = \varphi_j$ for all $j \neq i$. This, however, contradicts the fact that the considered execution has maximal response time. Note that shifting the release of task τ_i does not affect the higher priority tasks, and hence does not affect the start of the considered execution of τ_i.

(ii) From property (i) we derive that the release of the considered execution falls in either a level-($i-1$) busy interval $[a', f')$ or a level-($i-1$) idle interval $[f', a')$ with $a \neq f'$, where f' and a' denote a completion time and a release time of higher priority tasks, respectively. Let the release of the considered execution fall in a level-($i-1$) busy interval. Similarly to the proof of property (i), we can prove by means of a contradiction argument that a maximal response time can only be found when $a = a'$, i.e. that the release coincides with the release of a higher priority task. Hence, the release of the considered execution is immediately preceded by a non-empty sub interval of a level-($i-1$) idle interval. □

Lemma 4.2. *If an execution of task τ_i, which is released at time a and completed at time f, has maximal response time, then it is preempted only by releases of higher priority tasks τ_j that fall inside the interval $[a, f)$, and each of these releases preempts the execution of τ_i by an amount WC_j.*

Proof. Consider the preemption of the worst-case execution of τ_i by a higher priority task τ_j. Obviously, releases of τ_j at or after time f are too late to cause any preemption to the considered execution of τ_i. Next, releases of τ_j before time a do not cause any preemption of the considered execution of τ_i either. The reason for this is the fact that a non-empty sub interval of a level-$(i-1)$ idle interval ends at the release of τ_i, according to Lemma 4.1, so there cannot be any work pending of previous releases of τ_j. Hence we can conclude that only the releases at time a and strictly between time a and time f cause preemptions to the considered worst-case execution of τ_i. Furthermore, as these higher priority executions need to be finished before task τ_i can be completed, we know that each of the releases of τ_j between a and f preempts τ_i for an amount WC_j. □

Theorem 4.1. *In order to have a maximal response time for an execution k of task τ_i, i.e. to have $f_{ik} - a_{ik} = WR_i$, we may assume without loss of generality that the phasing φ is such that $\varphi_j = a_{ik}$ for all $j < i$. In other words, the phasing of the tasks' release times is such that the release of the considered execution of τ_i coincides with the simultaneous release for all higher priority tasks. This latter point in time is called a* critical instant *for task τ_i.*

Proof. We first determine the amount of preemption of execution k of τ_i by higher priority tasks. Execution k of τ_i is released at time a_{ik} and completed at time f_{ik}, as indicated in Figure 4.2. Without loss of generality we assume that release number one of each higher priority task τ_j is the first release that takes place at or after time a_{ik}, i.e. we assume $a_{ik} \leq \varphi_j < a_{ik} + T_j$. This is indicated in the figure, where we numbered the releases of τ_j.

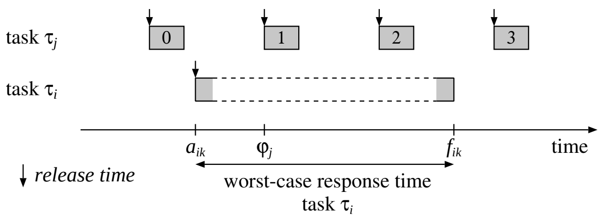


Figure 4.2. An execution k of τ_i with maximal response time, released at time a_{ik} and completed at time f_{ik}, and a higher priority task τ_j for which release one, taking place at time $a_{j1} = \varphi_j$, is the first release at or after time a_{ik}. The executions of τ_j are numbered for ease of reference.

According to Lemma 4.2, we can determine the amount of preemption of task τ_i by task τ_j, by counting the number of releases of τ_j in the interval $[a_{ik}, f_{ik})$. Given the numbering of the releases of τ_j as indicated before, this is given by the maximal number $m \in \mathbb{N}$ for which

$$\varphi_j + (m-1)T_j < f_{ik},$$

or, equivalently,

$$m - 1 < \frac{f_{ik} - \varphi_j}{T_j}.$$

The maximal number $m \in \mathbb{N}$ satisfying this inequality is given by

$$m = \left\lceil \frac{f_{ik} - \varphi_j}{T_j} \right\rceil.$$

As mentioned in Lemma 4.2, each of these releases gives a preemption of WC_j. The total amount of preemption of the considered worst-case execution of τ_i by higher priority tasks is thus given by

$$\sum_{j<i} \left\lceil \frac{f_{ik} - \varphi_j}{T_j} \right\rceil WC_j, \tag{4.1}$$

in which, as assumed, each φ_j is bounded by $a_{ik} \leq \varphi_j < a_{ik} + T_j$. Obviously, (4.1) is maximal when each φ_j is chosen minimally, i.e. $\varphi_j = a_{ik}$, which exactly proves the theorem. □

We can draw the following conclusion from Theorem 4.1.

Corollary 4.1. *The highest amount of preemption of a considered task is found right after a simultaneous release of higher priority tasks.* □

4.2 Response times

Based on a critical instant, we first show how to determine the worst-case response time by construction using a time line. Next, we derive a recursive equation for the worst-case response times and present an iterative procedure to find its solution. Finally, we illustrate the iterative procedure by an example. As example, we will use the task set with the characteristics given in Table 3.1.

4.2.1 A time line

A task τ_i can execute if and only if no task with a higher priority is executing. Hence, the execution of task τ_i fills the level-$(i-1)$ idle intervals. Given Theorem 4.1, we therefore draw the following conclusion.

Corollary 4.2. *The worst-case response time WR_i of task τ_i can be determined by construction using a time line with a simultaneous release of τ_i with all higher priority tasks at time a by the following iterative procedure.*

(i) Draw a time line with releases of the highest priority task τ_1, with a first release at time a. For every release, task τ_1 can start immediately and execute to completion.

(ii) Given the responses of tasks τ_1 till τ_{i-1}, let task τ_i have a first release at time a. The execution of task τ_i fills the level-$(i-1)$ idle intervals till τ_i has executed its computation time.

The procedure is stopped when either the response time of a task τ_j with $j \le i$ exceeds its deadline D_j, in which case the set of tasks is not schedulable, or when τ_i completes. In the latter case, the worst-case response time WR_i is found. □

Figure 4.3 illustrates the procedure for our example. From the figure, we derive that $WR_1 = 3$, $WR_2 = 17$, and $WR_3 = 56$.

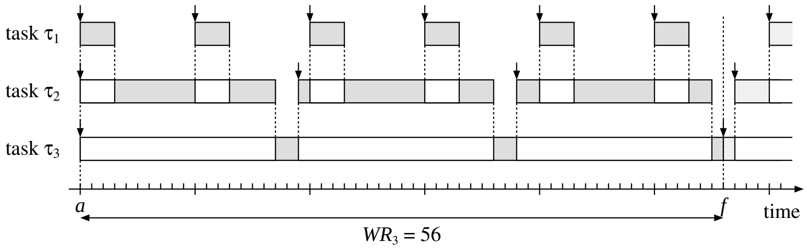


Figure 4.3. The constructed worst-case response time of task τ_3.

4.2.2 A recursive equation

We first observe that when the response time of a job of a task exceeds the period, that job also preempts the next job. As mentioned before, we only consider cases in this thesis where the worst-case response times are less than or equal to periods. This gives rise to a precondition in the following theorem.

Theorem 4.2. *The worst-case response time WR_i of a task τ_i is given by the smallest $x \in \mathbb{R}^+$ that satisfies the following equation, provided that x is at most T_i.*

$$x = WC_i + \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j. \tag{4.2}$$

Proof. The worst-case response time of τ_i consists of two parts: its computation time WC_i and its preemption by higher priority tasks. The latter is given by (4.1), in which we may substitute φ_j by a_{ik} according to Theorem 4.1. Together with the fact that $f_{ik} - a_{ik} = WR_i$, this shows that $x = WR_i$ indeed satisfies (4.2).

To show that WR_i is the smallest positive value that satisfies the recursive equation, we show that any $x \in \mathbb{R}^+$ that satisfies it, is an upper bound on WR_i. To this end, consider an interval of length x that starts at time a_{ik}, the time at which the considered worst-case execution k of τ_i is released. Then we know that in this interval *at most* an amount

$$\sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j$$

of preemption takes place by higher priority tasks. Therefore, *at least* an amount

$$x - \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j = WC_i$$

remains for executing task τ_i, which is at worst just enough. Hence, the corresponding completion of τ_i must take place at time $a_{ik} + x$ or before, and hence x is an upper bound on WR_i. □

We mention that there may be multiple values that satisfy (4.2), as is the case for our example, where both the values $x = 56$ and $x = 73$ satisfy (4.2).

4.2.3 An iterative procedure

As mentioned before, we assume deadlines to be hard, i.e. $WR_i \leq D_i$ (3.7). This assumption is therefore also used as a termination criterion in the next theorem.

Theorem 4.3. *The worst-case response time WR_i of task τ_i can be found by the following iterative procedure.*

$$WR_i^{(0)} = WC_i \tag{4.3}$$

$$WR_i^{(l+1)} = WC_i + \sum_{j<i} \left\lceil \frac{WR_i^{(l)}}{T_j} \right\rceil WC_j, \quad l = 0, 1, \ldots \tag{4.4}$$

The procedure is stopped when the same value is found for two successive iterations of l, or when the deadline D_i is exceeded.

Proof. We first prove termination of the procedure, by showing that the sequence is non-decreasing and that it can only take on a finite number of values, and hence eventually either two successive iterations must give the same value or the deadline D_i is exceeded. When the procedure stops because the same value is found for two successive iterations, we know that we have a solution of (4.2). To show that the found value is indeed WR_i, i.e. the smallest positive solution of (4.2), we also show that all values in the sequence $WR_i^{(l)}$ are lower bounds on WR_i.

We first show that the sequence is non-decreasing, by induction. To this end, we start by noting that $WR_i^{(0)} = WC_i > 0$, and

$$WR_i^{(1)} = WC_i + \sum_{j<i} \left\lceil \frac{WR_i^{(0)}}{T_j} \right\rceil WC_j \geq WC_i = WR_i^{(0)}.$$

Next, if $WR_i^{(l+1)} \geq WR_i^{(l)}$, then we can conclude from (4.4) that also $WR_i^{(l+2)} \geq WR_i^{(l+1)}$, as filling in a higher value in the right-hand side of (4.4) gives a higher or equal result.

Next, we prove that the sequence can only take on a finite number of values. To this end, we note that $WR_i^{(l)}$ is bounded from below by WC_i and from above by D_i. This means that each factor

$$\left\lceil \frac{WR_i^{(l)}}{T_j} \right\rceil$$

in (4.4) can only take on a finite number of values. Combining this for all higher priority tasks τ_j, we can conclude that the right-hand side of (4.4) can only take on a finite number of values.

We finally prove $WR_i^{(l)} \leq WR_i$, for all $l = 0, 1, \ldots$, by induction. Obviously, $WR_i^{(0)} = WC_i \leq WR_i$. Next, if $WR_i^{(l)}$ is a lower bound on WR_i, then

$$\sum_{j<i} \left\lceil \frac{WR_i^{(l)}}{T_j} \right\rceil WC_j$$

is a lower bound on the amount of preemption of the worst-case execution of τ_i by higher priority tasks, and hence $WR_i^{(l+1)}$ is also a lower bound on WR_i. □

All tasks with a higher priority than τ_i preempt τ_i at least once. Therefore, $\sum_{j \leq i} WC_j$ is typically taken as the initial value for the iterative procedure to calculate WR_i [Audsley et al., 1991; Joseph and Pandya, 1986; Klein et al., 1993]. Without further elaboration we mention that for $i > 1$, $WR_{i-1} + WC_i$ is also a lower bound on the worst-case response time WR_i, so we can use this value too for initialization.

4.2.4 An example

To illustrate the iterative procedure for worst-case response times, consider the example of Figure 3.3, of which the characteristics are given in Table 3.1. Figure 4.4 shows the successive iterations of the computation of the worst-case response time of task τ_3, using $\sum_{j \leq 3} WC_j$ as initial value, and yielding an eventual value of 56. As we can see in this figure, the initial lower bound $WR_3^{(0)} = 19$, indicated by a dashed line, is also preempted by execution 2 of τ_1. So the preemptions contain at least executions 1 and 2 of τ_1 and execution 1 of τ_2. As a result, we can conclude that this execution preempts a worst-case execution of τ_3, resulting in a new upper bound of $WR_3^{(1)} = 5 + 2 \cdot 3 + 1 \cdot 11 = 22$.

Next, in iteration (ii), we see that $WR_3^{(1)}$ falls inside the period of execution 3 of τ_1 and execution 2 of τ_2, so the preemptions at least contain executions 1,

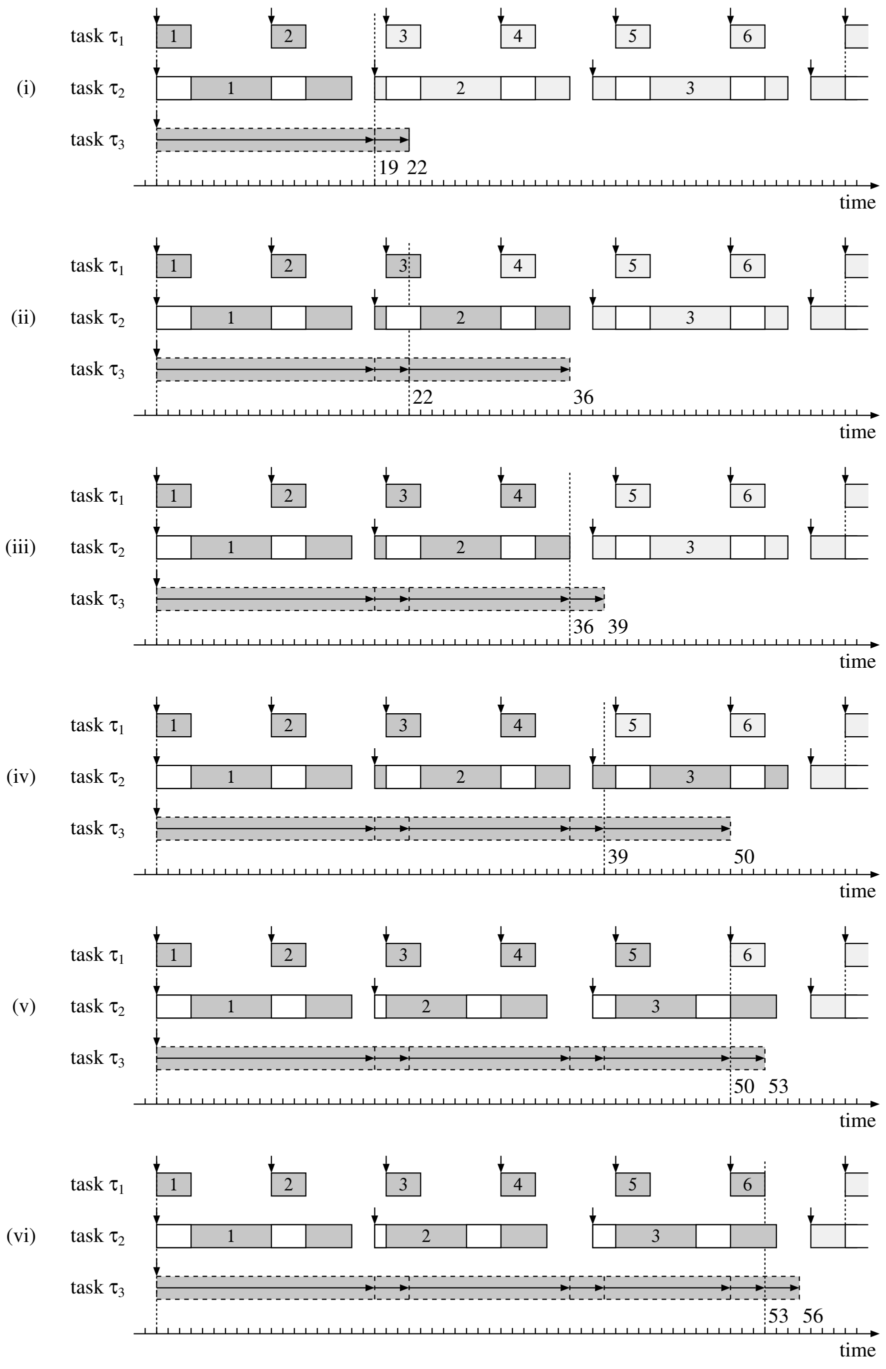


Figure 4.4. Iterations to determine the worst-case response time of task τ_3. Executions are again numbered for ease of reference.

2, and 3 of τ_1 and executions 1 and 2 of τ_2, resulting in a new upper bound of $WR_3^{(2)} = 5+3\cdot3+2\cdot11 = 36$.

This continues up to iteration (vi). There, we see that executions $1,\ldots,6$ of τ_1 and executions $1,\ldots,3$ of τ_2 preempt the worst-case execution of τ_3, resulting in $WR_3^{(6)} = 5+6\cdot3+3\cdot11 = 56$. As $WR_3^{(6)} = WR_3^{(7)}$, the procedure stops. The worst-case response time is therefore 56, of which the execution and preemptions were already shown in Figure 4.3.

Note that time lines are an easy-to-understand graphical representation for the recursive equation and its underlying notion of critical instant. However, the recursive equation is more appropriate for calculation purposes.

4.3 Occupied times

In Figure 4.3, task τ_3 is preempted at time 19 due to a release of τ_2, and resumes execution at time 36. The span of time from a task τ's release till the moment in time that τ can start or resume its execution after completion of a computation C is termed the *occupied time*. The *worst-case occupied time* (WO) of a task τ is the longest possible span of time from a release of τ till the moment in time that τ can start or resume its execution after completion of a computation C. For a computation time $C > 0$, that next moment is identical to the worst-case response time WR if and only if no higher priority task is released at that moment. Otherwise, executions of higher priority tasks aligned at WR will delay the resumption.

Figure 4.5 illustrates the worst-case response time WR_3 and worst-case occupied time WO_3 of task tau_3 as functions of the computation time C_3. Note that the difference between both notions is in the (open or closed) end-points of the line fragments. Further note that WR_3 is only defined for positive computation times, whereas WO_3 is also defined for a computation time of zero. For a computation time $C = 0$, the worst-case occupied time is equal to the worst-case start time, i.e.

$$WS_i = WO_i(0) \tag{4.5}$$

In the remainder of this document, we will therefore treat the worst-case start time as a special case of the worst-case occupied time.

We will first determine the worst-case occupied time of a task τ by construction using a time line with a critical instant for τ. A next theorem presents a recursive equation for the worst-case occupied time of a task. That equation is derived from the recursive equation (4.2) for the worst-case response time, and uses specific properties of (4.2) and its solutions. We therefore revisit worst-case response times before presenting the theorem.

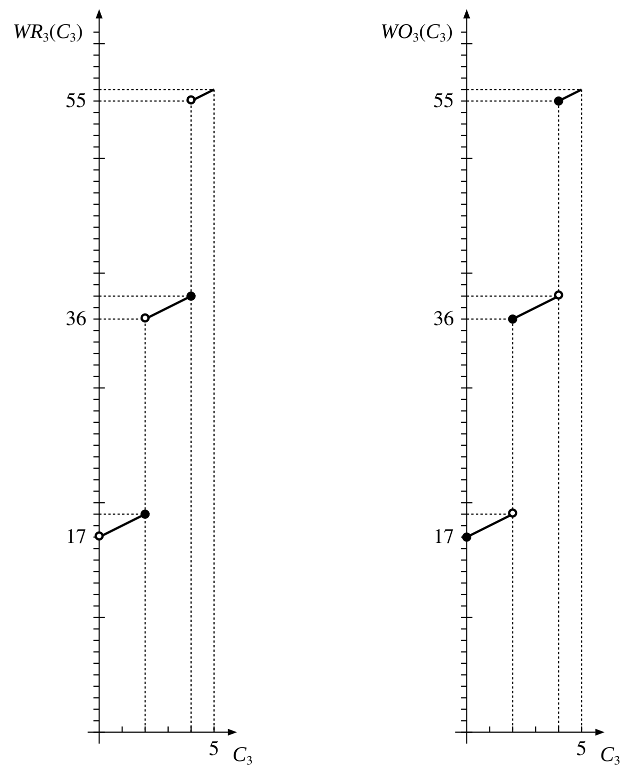


Figure 4.5. The worst-case response time WR_3 and worst-case occupied time WO_3 of task τ_3 as functions of the computation time C_3.

4.3.1 A time line

Considering Figure 4.3, τ_3 can start its execution at time 17, therefore $WS_3 = WO_3(0) = 17$. As mentioned above, task τ_3 is preempted by task τ_2 at time 19 after an execution of a computation time of 2, and can resume its execution at time 36. Hence, $WR_3(2) = 19 < WO_3(2) = 36$. Task τ_3 can continue its execution after a computation time of 1 at time 18, and the worst-case occupied time and the worst-case response time of τ_3 are therefore the same for a computation time $C_3 = 1$, i.e. $WO_3(1) = WR_3(1) = 18$.

4.3.2 Response time revisited

In Section 4.2, we did not determine under which conditions there actually exists a positive solution for (4.2). Instead, we used the exact schedulability condition (3.7) as a termination condition in Theorem 4.3 for the iterative procedure to determine the worst-case response time. In this section, we will take a closer look at (4.2) and its solutions.

In general, there remains processor time for execution of τ_i when the cumulative utilization factor U_{i-1} of the tasks with a higher priority than τ_i is less than one. Hence, assuming only a single job for τ_i, there will eventually be sufficient time for τ_i to execute any finite computation time $C_i > 0$. Strictly spoken, the smallest positive solution of (4.2) is the response time r_i of τ_i that is released simultaneously with all higher priority periodic tasks taking only *a single job* of τ_i in consideration. Intuitively, there therefore exists a smallest solution for (4.2) when $U_{i-1} < 1$.

A next lemma states that there exists a positive solution for (4.2) if and only if $U_{i-1} < 1$. As a consequence, when the necessary condition for the schedulability of a set $\mathcal{T}$ of n periodic tasks as expressed by (3.11) holds, then there exist positive solutions for (4.2) for all tasks. For the proof of that lemma we use a variant of Bolzano's theorem for continuous functions (see, for example, [Harris & Stocker, 1998]).

Bolzano's theorem. *Let, for two real a and b, $a < b$, a function f be continuous on a closed interval $[a, b]$ such that $f(a)$ and $f(b)$ are of opposite signs. Then there exists a number $c \in [a, b]$ such that $f(c) = 0$.* □

For a continuous function $f(x)$, the next lemma can be easily proven by introducing a new function $g(x) = f(x) - x$. In our variant, the function $f(x)$ is assumed to be defined and strictly non-decreasing, and we therefore need an alternative proof.

Lemma 4.3. *Let $f(x)$ be defined and strictly non-decreasing in an interval $[a, b]$ with $f(a) > a$ and $f(b) < b$. Then there exists a value $c \in (a, b)$ such that $f(c) = c$.*

Proof. Figure 4.6 illustrates the lemma. Let c be defined by

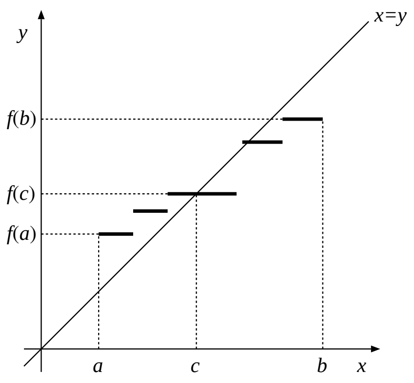


Figure 4.6. A defined and strictly non-decreasing function $f(x)$ on $[a, b]$.

$$c = \sup\left\{x \geq a | \forall_{t \in [a,x]} f(t) \geq t\right\}.$$

This supremum exists, because the set is bounded, i.e.

$$\underset{t\in[a,x]}{\forall} f(t) \geq t \Rightarrow x < b$$

due to $f(b) < b$, and therefore $c \leq b$. We will now prove $f(c) = c$ by means of a contradiction argument, i.e. we show that both assumptions $f(c) > c$ and $f(c) < c$ lead to contradictions.

Assume $f(c) > c$. Then choose $x = c + (f(c) - c)/2 = (f(c) + c)/2 > c$. For $t \in [a,x]$, we distinguish two cases. For $t < c$, $f(t) \geq t$ by the definition of c. For $c \leq t \leq x$, we get:

$$\begin{array}{rcl} f(t) & \geq & \{\text{by definition of } f\} \;\; f(c) \\ & > & \{\text{by our assumption } f(c) > c\} \;\; (f(c)+c)/2 \\ & = & x \geq t. \end{array}$$

Moreover, x is at most b. This is because $c \leq b$, as a result we get $f(c) \leq f(b) < b$, and therefore $x = (f(c)+c)/2 < (b+b)/2 = b$. Hence, for $t \in [a,x]$ we get $f(t) \geq t$. From the definition of c, we now derive $c \geq x$, which contradicts the fact that $x > c$. Hence, the assumption $f(c) > c$ was wrong.

Assume $f(c) < c$. Then choose $t = (f(c) + c)/2 < c$. We now get:

$$\begin{array}{rcl} f(t) & \leq & \{\text{by definition of } f\} \;\; f(c) \\ & < & \{\text{by our assumption } f(c) < c\} \;\; (f(c)+c)/2 \\ & = & t. \end{array}$$

Moreover, this value for t is at least a. This is because $c \geq a$, as a result we get $f(c) \geq f(a) > a$, and therefore $t = (f(c)+c)/2 > (a+a)/2 = a$. Hence, when x satisfies

$$\underset{t'\in[a,x]}{\forall} f(t') \geq t',$$

x must be smaller than t, i.e. $x < (f(c)+c)/2$. Given the definition of c (supremum), we get

$$\begin{array}{rcl} c & = & \sup x \geq a | \forall_{t'\in[a,x]} f(t') \geq t' \\ & \leq & \sup x \geq a | x < (f(c)+c)/2 \\ & = & \dfrac{f(c)+c}{2} < c, \end{array}$$

which again gives a contradiction. Hence, our assumption $f(c) < c$ was wrong.

We now have $c \geq a$, $c \leq b$, and $f(c) = c$. From $f(a) > a$ and $f(b) < b$, we conclude that $c \neq b$ and $c \neq a$, i.e. $c \in (a,b)$. □

Lemma 4.4. *There exists a positive solution for the recursive equation (4.2) for the worst-case response time WR_i of a task τ_i if and only if $U_{i-1} < 1$.*

Proof. For $i = 1$, the lemma trivially holds. Assume $i > 1$, and let f be defined as

$$f(x) = WC_i + \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j.$$

$\{$*only if* $U_{i-1} < 1\}$ The condition is necessary, as for its negation $U_{i-1} \geq 1$ we can derive

$$f(x) \geq WC_i + \sum_{j<i} \frac{x}{T_j} WC_j = WC_i + xU_{i-1} > x$$

for all $x > 0$.

$\{$*if* $U_{i-1} < 1\}$ We will prove that the condition is sufficient by means of Lemma 4.3. Let $f(t)$ be as defined above. We choose $a = 0$, hence $f(a) = f(0) = WC_i > 0 = a$. In order to choose an appropriate b, we make a derivation similar to the one given above.

$$f(x) < WC_i + \sum_{j<1} \left(\frac{x}{T_j} + 1 \right) WC_j = \sum_{j \leq i} WC_j + xU_{i-1}$$

The relation $x \geq \sum_{j \leq i} WC_j + xU_{i-1}$ holds for $x \geq \sum_{j \leq i} WC_j / (1 - U_{i-1})$ as $U_{i-1} < 1$. We now choose $b = \sum_{j \leq i} WC_j / (1 - U_{i-1})$, and therefore get $b > f(b)$. Now the conditions for Lemma 4.3 hold, i.e. the function $f(x)$ is defined and strictly non-decreasing in an interval $[a, b]$ with $f(a) > a$ and $f(b) < b$. Hence, there exists an $x \in (0, \sum_{j \leq i} WC_j / (1 - U_{i-1}))$ such that $x = f(x)$. □

From the construction of the worst-case response time WR_i of task τ_i using a time line as described by Corollary 4.2 and the fact that time lines are a graphical representation for the recursive equation (4.2) and its underlying notion of critical instant, we draw the following intuitive conclusion.

Corollary 4.3. *For $U_{i-1} < 1$ and $C_i > 0$, the smallest positive solution $WR_i(C_i)$ of (4.2) has the following properties.*

(i) $WR_i(C_i)$ is a defined and strictly increasing function of C_i.

(ii) $WR_i(C_i)$ is discontinuous at points C_i where $WR_i(C_i) = kT_j$, with $k \in \mathbb{Z}^+$ and $j < i$.

(iii) $WR_i(C_i)$ is left-continuous, i.e.

$$\lim_{x \uparrow C_i} WR_i(x) = WR_i(C_i). \tag{4.6}$$

□

4.3.3 A recursive equation

From the description of the worst-case occupied time $WO_i(C_i)$ of a task τ_i with computation time $C_i \geq 0$ and its construction by means of a time line we derive

$$WO_i(C_i) = \lim_{x \downarrow C_i} WR_i(x). \tag{4.7}$$

We will use (4.7) as a starting point for the derivation of the recursive equation for the worst-case occupied time. First we prove the following lemma.

Lemma 4.5. *When* $\lim_{x \downarrow X} f(x)$ *is defined, and* $f(x)$ *is strictly increasing in an interval* $(X, X + \gamma)$ *for sufficiently small* $\gamma \in \mathbb{R}^+$*, then the following equation holds.*

$$\lim_{x \downarrow X} \lceil f(x) \rceil = \left\lfloor \lim_{x \downarrow X} f(x) \right\rfloor + 1 \tag{4.8}$$

Proof. The proof uses the definition of limit:

$$\lim_{x \downarrow X} f(x) = Y \Leftrightarrow \underset{\varepsilon>0}{\forall} \ \underset{\delta>0}{\exists} \ \underset{x \in (X, X+\delta)}{\forall} |f(x) - Y| < \varepsilon.$$

We first prove the relation

$$\underset{X<x<X+\gamma}{\forall} f(x) > Y,$$

and subsequently prove the lemma.

The proof of the relation is based on a contradiction argument. Because $\lim_{x \downarrow X} f(x)$ is defined, we may write $\lim_{x \downarrow X} f(x) = Y$. Assume $f(x_1) \leq Y$ for an $x_1 \in (X, X + \gamma)$. Choose an $x_2 \in (X, x_1)$. Because $f(x)$ is strictly increasing in $(X, X + \gamma)$, $f(x_2) < f(x_1) \leq Y$. Now choose $\varepsilon = Y - f(x_2)$, then

$$\forall_{x \in (X, x_2)} f(x) < f(x_2) < Y$$

and hence

$$|f(x) - Y| > |f(x_2) - Y| = \varepsilon,$$

which contradicts the fact that $\lim_{x \downarrow X} f(x) = Y$.

For the proof of the lemma, we consider two main cases: $Y \in \mathbb{Z}$ and $Y \notin \mathbb{Z}$. Let $Y \in \mathbb{Z}$. According to the relation proven above, $0 < f(x) - Y$ for all $x \in (X, X + \gamma)$. Let $\varepsilon \in (0, 1]$. Now there exists a $\delta_1 \in (0, \gamma)$ such that $0 < f(x) - Y < \varepsilon \leq 1$ for all $x \in (X, X + \delta_1)$, hence $Y < f(x) < Y + 1$, i.e. $\lceil f(x) \rceil = Y + 1 = \lfloor Y \rfloor + 1$. So,

$$\lim_{x \downarrow X} \lceil f(x) \rceil = \lim_{x \downarrow X} (\lfloor Y \rfloor + 1) = \lfloor Y \rfloor + 1 = \left\lfloor \lim_{x \downarrow X} f(x) \right\rfloor + 1.$$

Next, let $Y \notin \mathbb{Z}$. Let $\varepsilon \in (0, \lceil Y \rceil - Y]$. Now there exists a $\delta_2 \in (0, \gamma)$ such that for all $x \in (X, X + \delta_2)$

$$0 < f(x) - Y < \varepsilon \leq \lceil Y \rceil - Y,$$

hence

$$Y < f(x) < Y + \varepsilon < \lceil Y \rceil,$$

i.e.

$$\lceil f(x) \rceil = \lceil Y \rceil = \lfloor Y \rfloor + 1.$$

For this second main case we therefore also find

$$\lim_{x \downarrow X} \lceil f(x) \rceil = \lim_{x \downarrow X} (\lfloor Y \rfloor + 1) = \lfloor Y \rfloor + 1 = \left\lfloor \lim_{x \downarrow X} f(x) \right\rfloor + 1,$$

which proves the lemma. □

Theorem 4.4. *When the smallest positive solution WR_i of (4.2) for a computation time $WC_i^!$ is at most $\min(D_i, T_i)$, the worst-case occupied time WO_i of a task τ_i with a computation time $WC_i \in [0, WC_i^!]$ is given by the smallest non-negative $x \in \mathbb{R}$ that satisfies*

$$x = WC_i + \sum_{j<i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor + 1 \right) WC_j. \tag{4.9}$$

Proof. Given Lemma 4.5, we can make the following derivation starting from (4.7).

$$\begin{aligned}
WO_i(WC_i) &= \lim_{x \downarrow WC_i} WR_i(x) \\
&= \{(4.2)\} \lim_{x \downarrow WC_i} \left(x + \sum_{j<i} \left\lceil \frac{WR_i(x)}{T_j} \right\rceil WC_j \right) \\
&= WC_i + \sum_{j<i} \lim_{x \downarrow WC_i} \left\lceil \frac{WR_i(x)}{T_j} \right\rceil WC_j \\
&= \{\text{Lemma 4.5}\} WC_i + \sum_{j<i} \left(\left\lfloor \lim_{x \downarrow WC_i} \frac{WR_i(x)}{T_j} \right\rfloor + 1 \right) WC_j \\
&= \{(4.7)\} WC_i + \sum_{j<i} \left(\left\lfloor \frac{WO_i(WC_i)}{T_j} \right\rfloor + 1 \right) WC_j
\end{aligned}$$

Remember that the smallest positive solution of (4.2) only consider a single job of the task under consideration. The same holds for the smallest non-negative solution of (4.9). Hence, the worst-case occupied time WO_i of task τ_i with a computation time $WC_i \in [0, WC_i^!]$ is the smallest non-negative solution WO_i satisfying (4.9), when the smallest positive solution WR_i of (4.2) for a computation time $WC_i^!$ is at most $\min(D_i, T_i)$. □

Note that Theorem 4.4 has a precondition, similar to Theorem 4.2 for the worst-case response time of a task.

4.3.4 An iterative procedure

The next theorem describes how the smallest positive solution of (4.4) can be found using an iterative procedure.

First, we prove the following lemma.

Lemma 4.6. *There exists a non-negative solution for the recursive equation (4.9) for the worst-case occupied time WO_i of a task τ_i if $U_{i-1} < 1$.*

Proof. For $i = 1$, the lemma trivially holds. For $i > 1$, the proof is similar to the $\{U_{i-1} < 1\}$ part of the proof of Lemma 4.4, i.e. we prove that the condition is sufficient by means of Lemma 4.3. □

Theorem 4.5. *The worst-case occupied time WO_i of task τ_i can be found by the following iterative procedure.*

$$WO_i^{(0)} = \begin{cases} \sum_{j<i} WC_j & \text{for } WC_i = 0 \\ WR_i & \text{for } WC_i > 0 \end{cases} \tag{4.10}$$

$$WO_i^{(l+1)} = WC_i + \sum_{j<i} \left(\left\lfloor \frac{WO_i^{(l)}}{T_j} \right\rfloor + 1 \right) WC_j, \quad l = 0, 1, \ldots \tag{4.11}$$

The procedure is stopped when the same value is found for two successive iterations of l.

Proof. The proof is similar to that of Theorem 4.3. □

Note that by starting with $WO_i^{(0)} = WR_i$, we have to determine WR_i first. Given Lemma 4.4, it is therefore guaranteed that when WR_i exists, there also exists a solution for (4.9). In such a situation, Theorem 4.5 yields the worst-case occupied time WO_i. On the other hand, existence of a solution of (4.9) does not necessarily imply that it represents WO_i. As an example, when the smallest positive solution of (4.2) is larger than T_i the job under consideration also preempts the next job, and that situation has been ignored in (4.9).

Without further elaboration we mention that for $WC_i = 0$ and $i > 1$, WO_{i-1} is also a lower bound on the worst-case occupied time WO_i, so we can use this value too for initialization.

4.4 Discussion

In this section, we consider worst-case execution times and initial values for the iterative procedure to determine the worst-case response time, present several properties of the worst-case occupied time, and finally introduce the notion of *worst-case induced load.*

4.4.1 Execution times

Given (3.1) and the definition of the execution interval $e_{ik} = a_{ik} - s_{ik}$, we immediately derive $WE_i \leq WR_i$. The next lemma states that the worst-case execution time equals the worst-case response time.

Lemma 4.7. *The worst-case execution time WE_i of a task τ_i is equal to the worst-case response time WR_i of that task.*

Proof. The proof is by construction. Consider a simultaneous release of task τ_i and all its higher priority tasks (i.e. a critical instant of τ_i) at time $t = 0$. Let this release of τ_i be denoted by job k. There exists a resume time $t < f_{ik}$, such that all preemptions of tasks with a higher priority than τ_i are completed, and the last part of job k lasting a time x_ω can be executed without preemptions between time t and f_{ik}; see Figure 4.7. We now move the release of task τ_i an amount of

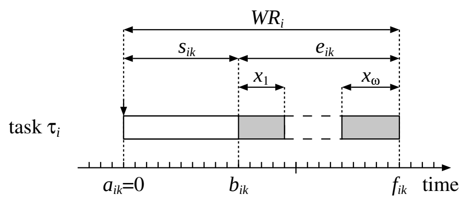


Figure 4.7. Release of task τ_i at a critical instant at time $t = 0$.

time $x_\omega/2$ backwards in time, i.e. $a'_{ik} = a_{ik} - x_\omega/2 = -x_\omega/2$. Because the response time of job k already was the worst-case response time WR_i, moving its release backwards in time cannot increase its response time. Hence, job k can immediately start executing at time $-x_\omega/2$, and the length of the start interval becomes zero, i.e. $s'_{ik} = b'_{ik} - a'_{ik} = 0$. The response time of k cannot decrease either, because an amount $x_\omega/2$ is still to be executed in the interval $[f_{ik} - x_\omega, f_{ik} - x_\omega/2)$, i.e. job k completes at $f'_{ik} = f_{ik} - x_\omega/2$. As a result, the length of the execution interval equals the response time, i.e. $e'_{ik} = f'_{ik} - b'_{ik} = f_{ik} - x_\omega/2 - (a_{ik} - x_\omega/2) = r_{ik}$. By keeping

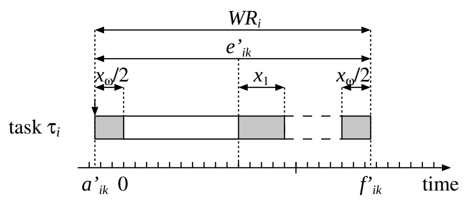


Figure 4.8. Release of task τ_i at $t = -x_\omega/2$ and a simultaneous release of all higher priority tasks at time $t = 0$.

the response time of job k the same, we therefore constructed an execution interval equal to the worst-case response time, so $WE_i \geq WR_i$; see Figure 4.8. Together with the relation $WE_i \leq WR_i$ this proves the lemma. □

From Corollary 4.2, we know that the execution of τ_i may be viewed to fill level-$(i-1)$ idle intervals. Assuming an arbitrary computation time C_i for τ_i, and therefore an x_ω with the length of an arbitrary level-$(i-1)$ idle interval, we may draw the following conclusion from the proof of Lemma 4.7.

Corollary 4.4. *A longest level-$(i-1)$ idle interval is found right before a simultaneous release of tasks τ_1 till τ_{i-1}.* □

Corollary 4.5. *The worst-case execution time WE_i of a task τ_i is not assumed when that task is simultaneously released with all its higher priority tasks. Instead, it is assumed when that task τ_i is released just before the simultaneous release of all its higher priority tasks.* □

For a critical instant, it is possible to define a *condition* (i.e. the simultaneous release of a task with all its higher priority tasks; see Theorem 4.1) and subsequently associate a *single moment in time* with that condition such that the worst-case response of a task is assumed when that task starts at that moment in time. The term *instant* in critical instant may therefore refer to both the condition as well as the moment in time for which the condition holds. Similarly to the worst-case response time, it is also possible to define a condition for the instant for which a worst-case execution time of a task is assumed. Unlike the worst-case response time, *two moments of time* are associated with that condition: the release of the task under consideration and the simultaneous release of all its higher priority tasks.

4.4.2 Initial values

In this section, we first consider standard initial values for the iterative procedure to determine the worst-case response time. Next, we present an alternative initial value. Finally, we propose an initial value that combines standard and alternative initial values.

Standard initial values

As mentioned before, $\sum_{j \leq i} WC_j$ is typically taken as the initial value for the iterative procedure to calculate WR_i. Moreover, $WR_{i-1} + WC_i$ is an attractive alternative when WR_{i-1} is known. In both cases, the initial value is a constant plus WC_i. We will therefore use the term $\iota_i^{\mathrm{SW}}(WC_i)$ to denote a standard initial value, which is defined using a constant χ_i^{W}.

$$\iota_i^{\mathrm{SW}}(WC_i) = \chi_i^{\mathrm{W}} + WC_i \tag{4.12}$$

In this document, we assume $\chi_i^{\mathrm{W}} = WR_{i-1}$.

Alternative initial value

The definition of the alternative initial value $\iota_i^{\mathrm{AW}}(WC_i)$ and its appropriateness as initial value are given in Lemma 4.8.

Lemma 4.8. *The value $\iota_i^{\mathrm{AW}}(WC_i)$ defined by*

$$\iota_i^{\mathrm{AW}}(WC_i) = \frac{WC_i}{(1 - WU_{i-1})}, \tag{4.13}$$

is an appropriate initial value for the iterative procedure to determine $WR_i(WC_i)$.

Proof. The worst-case response time $WR_i(WC_i)$ is the smallest positive solution of (4.2), and the iterative procedure therefore has to start with a lower bound. Hence, we have to prove that $\iota_i^{\mathrm{AW}}(WC_i)$ is a lower bound for $WR_i(WC_i)$. To this end, we derive

$$\begin{aligned} WR_i(WC_i) &= WC_i + \sum_{j<i} \left\lceil \frac{WR_i(WC_i)}{T_j} \right\rceil WC_j \\ &\geq WC_i + \sum_{j<i} \left(\frac{WR_i(WC_i)}{T_j} \right) WC_j = WC_i + WR_i(WC_i) WU_{i-1}. \end{aligned}$$

Hence for $WU_{i-1} < 1$, we get $WR_i(WC_i) \geq WC_i/(1 - WU_{i-1})$. □

Although we independently conceived this alternative initial value, the credits for it should go to Mikael Sjödin who already conceived it in 1998 [Sjödin & Hansson, 1998]. The initial value presented in that paper also takes blocking and jitter into account.

Combined initial value

The highest amount of preemptions in a schedule σ_{i-1} is found right after the simultaneous release of tasks τ_1 till τ_{i-1}; see also Corollary 4.1. Whereas the standard initial value $\iota_i^{\mathrm{SW}}(WC_i)$ takes this aspect into account by having an offset $\chi_i = WR_{i-1}$, the alternative initial value $\iota_i^{\mathrm{AW}}(WC_i)$ does not. As a consequence, $\iota_i^{\mathrm{SW}}(WC_i)$ may perform better for small computation times. We therefore propose to use an initial value that combines $\iota_i^{\mathrm{SW}}(WC_i)$ and $\iota_i^{\mathrm{AW}}(WC_i)$. This proposed initial value $\iota_i^{\mathrm{CW}}(WC_i)$ to calculate $WR_i(WC_i)$ is defined by

$$\iota_i^{\mathrm{CW}}(WC_i) = \max(\iota_i^{\mathrm{SW}}(WC_i), \iota_i^{\mathrm{AW}}(WC_i)). \tag{4.14}$$

4.4.3 Properties of the worst-case occupied time

Lemma 4.9. *The worst-case occupied time WO_i of task τ_i is not a multiple of any period T_j for $j < i$.*

Proof. The proof is based on a contradiction argument. Assume the iterative procedure in Theorem 4.5 terminates with $WO_i^{(l)} = WO_i^{(l+1)} = mT_j$ and $WO_i^{(l-1)} < mT_j$. Then at most m activations of task τ_j are taken into account to determine $WO_i^{(l)}$. During the next step, when determining $WO_i^{(l+1)}$, $m+1$ activations will be taken into account, hence $WO_i^{(l+1)} > WO_i^{(l)}$. This contradicts the assumption. □

Lemma 4.10. *The worst-case occupied time WO_i of τ_i is a solution of the recursive equation for the worst-case response time (4.2).*

Proof. From the fact that the worst-case response time WR_i is a strictly increasing function of the computation time and from (4.7), we conclude that $WO_i \geq WR_i$. Because WO_i is not a multiple of any period T_j for $j < i$ (see previous lemma), $WO_i/T_j \notin \mathbb{Z}$ for any $j < i$. As a consequence,

$$\left\lfloor \frac{WO_i}{T_j} \right\rfloor + 1 = \left\lceil \frac{WO_i}{T_j} \right\rceil \quad \text{for } j < i.$$

This proves that WO_i is a solution of the recursive equation for worst-case response time (4.2). □

From the construction of the worst-case occupied time WO_i of task τ_i using a time line we draw the following intuitive conclusion.

Corollary 4.6. *For $U_{i-1} < 1$ and $C_i \geq 0$, the smallest non-negative solution $WO_i(C_i)$ of (4.9) has the following properties:*

(i) $WO_i(C_i)$ is a defined and strictly increasing function of C_i;

(ii) $WO_i(C_i)$ is discontinuous at points C_i where $WR_i(C_i)$ is discontinuous;

(iii) $WO_i(C_i)$ is right-continuous, i.e.

$$\lim_{x \downarrow C_i} WO_i(x) = WO_i(C_i). \tag{4.15}$$

□

We observe that we may also express the worst-case response time WR_i in terms of the worst-case occupied time WO_i, i.e.

$$WR_i(C_i) = \lim_{x \uparrow C_i} WO_i(x). \tag{4.16}$$

Note that (4.7) and (4.16) are similar, but not equivalent, i.e. one does not necessarily imply the other. This is due to the fact that the functors $\lim_{x \uparrow X}$ and $\lim_{x \downarrow X}$ are not injective.

4.4.4 Induced load

The summation term in the recursive equation (4.2) for the worst-case response time is also termed the *worst-case induced load*

$$WL_{i-1}(x) = \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j \tag{4.17}$$

of tasks τ_1 till τ_{i-1} in a right-open interval of length x, e.g. $[t, t+x)$. The worst-case induced load is the *maximal* load induced by tasks with a higher priority than τ_i.

Similarly, the summation term in (4.9) for the worst-case occupied time is the worst-case induced load

$$WL^{+}_{i-1}(x) = \sum_{j<i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor + 1 \right) WC_j \tag{4.18}$$

of tasks τ_1 till τ_{i-1} in a closed interval of length x, e.g. $[t, t+x]$.

The notions of worst-case induced load $WL_i(x)$ and $WL^{+}_i(x)$ can also be found in [George et al., 1996; Hermant et al., 1996].

5

Best-Case Analysis

This chapter presents theorems for the main best-case notions introduced in Chapter 3, i.e. *optimal instant* in Section 5.1, *best-case response time* in Section 5.2, and *best-case start time* and its generalization *best-case occupied time* in Section 5.3.

To show the duality of the best-case notions and the worst-case notions, the structure of this chapter closely follows that of Chapter 4, and even the text is at many places similar. Moreover, for every Theorem 4.x in Chapter 4 we have a similar Theorem 5.x in this chapter.

5.1 An optimal instant

Similar to the notion of a critical instant, which is used to derive worst-case response times, the next theorem introduces the notion of an *optimal instant*, as depicted in Figure 5.1. The theorem states that if we want to determine the best-case response time under arbitrary phasings, it suffices to consider optimal instants only. This notion is used in the next section to derive a recursive equation for best-case response times. The best-case response time analysis is based on best-case computation times BC_i.

First, we prove the following two lemmas.

Lemma 5.1. *If an execution of task τ_i, which is released at a time a, has minimal response time, then the following two properties hold.*

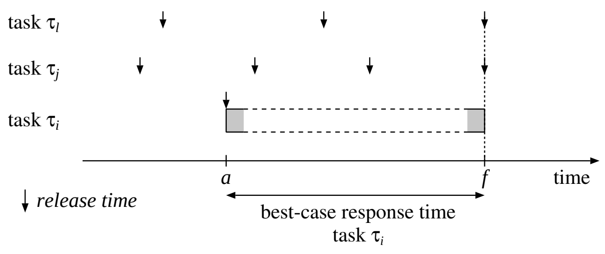


Figure 5.1. An optimal instant for task τ_i, at time f, where releases for all higher priority tasks (here τ_j and τ_l) coincide with the completion of an execution of τ_i.

(i) The execution starts right upon its release.
(ii) No higher priority task has a release at time a.

Proof. (i) If the considered execution starts at a time $b > a$, then we can decrease its response time by an amount $b - a$ by shifting its releases to the right by an amount $b - a$, i.e. by changing the phasing φ into a phasing φ' with $\varphi'_i = \varphi_i + b - a$ and $\varphi'_j = \varphi_j$ for all $j \neq i$. This, however, contradicts the fact that the considered execution has minimal response time. Note that shifting the releases of task τ_i does not affect the higher priority tasks, and hence does not affect the start of the considered execution of τ_i. Furthermore, note that the shifting cannot cause deadlines to be missed, as we assumed that each task's worst-case response time, i.e. the maximal response time under any phasing, is at most equal to the corresponding deadline.

(ii) If a higher priority task also has a release at time a, then this task is executed before task τ_i, and hence the considered execution of task τ_i starts at a time $b > a$. This, however, contradicts property (i). □

As a result of Lemma 5.1 and the notion of completion time, we know that task τ_i is being executed at both ends of the time interval between the release a and completion f of a best-case execution. This implies that all releases of higher priority tasks must have been completed. We use this property in the proof of the next lemma to determine the preemptions of a best-case execution.

Lemma 5.2. *If an execution of task τ_i, which is released (and started) at a time a and completed at time f, has minimal response time, then it is preempted only by releases of higher priority tasks τ_j that fall inside the open interval (a, f), and each of these releases preempts the execution of τ_i by an amount BC_j.*

Proof. Consider the preemption of the best-case execution of τ_i by a higher priority task τ_j. Obviously, releases of τ_j at or after time f are too late to cause any preemption to the considered execution of τ_i. Next, releases of τ_j before time a do not cause any preemption of the considered execution of τ_i either. The reason for this is the fact that τ_i starts executing right upon its release at time a, according to Lemma 5.1, so there cannot be any work pending of previous releases of τ_j. Next, Lemma 5.1 also tells us that there is no release of τ_j at time a, hence we can conclude that only the releases strictly between time a and time f cause preemptions to the considered best-case execution of τ_i. Furthermore, as these higher priority executions need to be finished before task τ_i can be completed, we know that each of the releases of τ_j between a and f preempts τ_i for an amount BC_j. □

Theorem 5.1. *In order to have a minimal response time for an execution k of task τ_i, i.e. to have $f_{ik} - a_{ik} = BR_i$, we may assume without loss of generality that the phasing φ is such that $\varphi_j = f_{ik}$ for all $j < i$. In other words, the phasing of the tasks' release times is such that the completion of the considered execution of τ_i coincides with a simultaneous release for all higher priority tasks. This latter point in time is called an* optimal instant *for task τ_i.*

Proof. We first determine the amount of preemption of execution k of τ_i by higher priority tasks. Execution k of τ_i is released at time a_{ik} and completed at time f_{ik}, as indicated in Figure 5.2. Without loss of generality we assume that release number zero of each higher priority task τ_j is the first release that takes place at or after time f_{ik}, i.e. we assume $f_{ik} \leq \varphi_j < f_{ik} + T_j$. This is indicated in the figure, where we numbered the releases of τ_j.

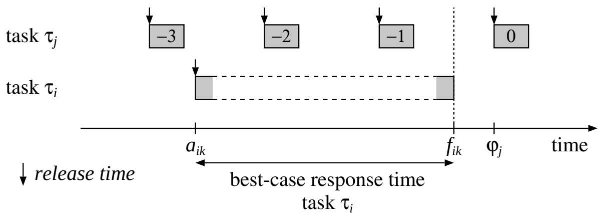


Figure 5.2. An execution k of τ_i with minimal response time, released at time a_{ik} and completed at time f_{ik}, and a higher priority task τ_j for which release zero, taking place at time $a_{j0} = \varphi_j$, is the first release at or after time f_{ik}. The executions of τ_j are numbered for ease of reference.

According to Lemma 5.2, we can determine the amount of preemption of task τ_i by task τ_j, by counting the number of releases of τ_j strictly between times a_{ik} and f_{ik}. Given the numbering of the releases of τ_j as indicated before, this is given by the maximal number $m \in \mathbb{N}$ for which

$$\varphi_j - mT_j > a_{ik},$$

or, equivalently,

$$m < \frac{\varphi_j - a_{ik}}{T_j}.$$

The maximal number $m \in \mathbb{N}$ satisfying this inequality is given by

$$m = \left\lceil \frac{\varphi_j - a_{ik}}{T_j} \right\rceil - 1.$$

As mentioned in Lemma 5.2, each of these releases gives a preemption of BC_j. The total amount of preemption of the considered best-case execution of τ_i by higher priority tasks is thus given by

$$\sum_{j<i} \left(\left\lceil \frac{\varphi_j - a_{ik}}{T_j} \right\rceil - 1 \right) BC_j, \tag{5.1}$$

in which, as assumed, each φ_j is bounded by $f_{ik} \leq \varphi_j < f_{ik} + T_j$. Obviously, (5.1) is minimal when each φ_j is chosen minimally, i.e. $\varphi_j = f_{ik}$, which exactly proves the theorem. □

We can draw the following conclusion from Theorem 5.1.

Corollary 5.1. *The lowest amount of preemption of a considered task is found right before a simultaneous release of higher priority tasks.* □

Remember that the notion of a critical instant tells us that right after such a moment the highest amount of preemption is found; see Corollary 4.1.

Note that whereas a critical instant gives the worst-case situation for all tasks simultaneously, an optimal instant gives the best case situation for one specific task. This implies that one particular phasing φ may give a best-case response time for one task, but another phasing φ' may have to be considered to obtain a best-case response time for another task.

5.2 Response times

Based on an optimal instant, we first show how to determine the best-case response time by construction using a time line. Next, we derive a recursive equation for the best-case response time and present an iterative procedure to find its solution. Finally, we illustrate the iterative procedure by an example.

5.2.1 A time line

The construction of the best-case response time BR_i of a task τ_i is similar to that of the worst-case response time WR_i as described in Corollary 4.2. The main differences being that (i) we have to make sure that the worst-case response time WR_i is smaller than the minimum of the deadline D_i and the period T_i, (ii) we draw the releases of the higher priority tasks *before* their simultaneous release, and (iii) we draw the response of τ_i *backwards* in time.

Corollary 5.2. *When the smallest positive solution of (4.2) is at most* $min(D_i, T_i)$*, the best-case response time* BR_i *of task* τ_i *can be determined by construction using a time line with a completion of* τ_i *that coincides with a simultaneous release for all higher priority tasks at time* f *by the following iterative procedure.*

(i) Draw a time line with releases of the highest priority task τ_1*, with computation times equal to* BR_1 *and a last release at time* $f - T_1$ *. For every release, task* τ_1 *can start immediately and execute to completion.*

(ii) Given the responses of tasks τ_1 *till* τ_{i-2}*, let task* τ_{i-1} *have a last release at time* $f - T_{i-1}$*. The first release of task* τ_{i-1} *should be* after *the first release of task* τ_{i-2} *to guarantee that all preemptions of the latter are taken into account for the response of task* τ_{i-1}*. The execution of task* τ_{i-1} *fills the level-(*$i-2$*) idle intervals till* τ_{i-1} *has executed its best-case computation time.*

(iii) The execution of task τ_i *fills the level-(*$i-1$*) idle interval starting from time* f backwards *in time till* τ_i *has executed its best-case computation time.*

The procedure is stopped when τ_i *completes, in which case the best-case response time* BR_i *is found.* □

Figure 5.3 illustrates the procedure for our example. Note that the characteristics of the example are given in Table 3.1. From the figure, we derive $BR_3 = 22$.

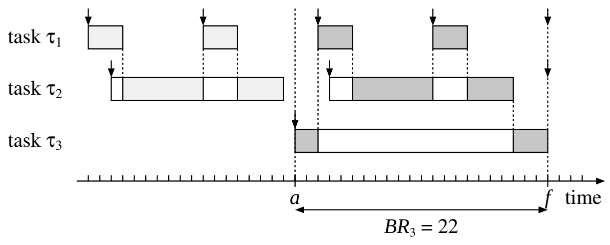


Figure 5.3. The constructed best-case response time of task τ_3.

5.2.2 A recursive equation

Next, we derive a recursive equation for the best-case response times, based on an optimal instant. Note that the following theorem also has a precondition, just like its dual Theorem 4.2 for the worst-case response time, to make sure that a previous job of τ_i cannot preempt the response under consideration.

Theorem 5.2. *When the smallest solution of (4.2) of a task* τ_i *is at most* $min(D_i, T_i)$*,* *the best-case response time BR_i of τ_i is given by the largest $x \in \mathbb{R}^+$ that satisfies*

$$x = BC_i + \sum_{j<i} \left(\left\lceil \frac{x}{T_j} \right\rceil - 1 \right) BC_j. \tag{5.2}$$

Proof. The best-case response time of τ_i consists of two parts: its computation time BC_i and its preemption by higher priority tasks. The latter is given by (5.1), in which we may substitute φ_j by f_{ik} according to Theorem 5.1. Together with the fact that $f_{ik} - a_{ik} = BR_i$, this shows that $x = BR_i$ indeed satisfies (5.2).

To show that BR_i is the largest value that satisfies the recursive equation, we show that any $x \in \mathbb{R}^+$ that satisfies it, is a lower bound on BR_i. To this end, consider an interval of length x that ends at time f_{ik}, the time at which the considered best-case execution k of τ_i completes. Then we know that in this interval *at least* an amount

$$\sum_{j<i} \left(\left\lceil \frac{x}{T_j} \right\rceil - 1 \right) BC_j$$

of preemption takes place by higher priority tasks. Therefore, *at most* an amount

$$x - \sum_{j<i} \left(\left\lceil \frac{x}{T_j} \right\rceil - 1 \right) BC_j = BC_i$$

remains for executing task τ_i, which is at best just enough. Furthermore, the counted preemptions are all due to releases of higher priority tasks strictly after time $f_{ik} - x$, so τ_i can only be completed in the interval $[f_{ik} - x, f_{ik})$ if it is also executed from time $f_{ik} - x$ to the first preemption after $f_{ik} - x$. Hence, the corresponding release of τ_i must take place at time $f_{ik} - x$ or before, and hence x is a lower bound on BR_i. □

We mention that there may be multiple values that satisfy (5.2), as is the case in the example of Section 5.2.4, where both the values $x = 22$ and $x = 5$ satisfy (5.2).

5.2.3 An iterative procedure

Theorem 5.3. *The best-case response time BR_i of task τ_i can be found by the following iterative procedure, which stops when the same value is found for two successive iterations of l.*

$$BR_i^{(0)} = WR_i \tag{5.3}$$

$$BR_i^{(l+1)} = BC_i + \sum_{j<i} \left(\left\lceil \frac{BR_i^{(l)}}{T_j} \right\rceil - 1 \right) BC_j, \quad l = 0, 1, \ldots \tag{5.4}$$

Proof. We first prove termination of the procedure, by showing that the sequence is non-increasing and that it can only take on a finite number of values, and hence eventually two successive iterations must give the same value. Upon termination, we know that we have a solution of (5.2). To show that the found value is indeed BR_i, i.e. the largest solution of (5.2), we show that all values in the sequence $BR_i^{(l)}$ are upper bounds on BR_i.

We first show that the sequence is non-increasing, by induction. To this end, we start by noting that $BR_i^{(0)} = WR_i$, and

$$\begin{aligned} BR_i^{(1)} &= BC_i + \sum_{j<i} \left(\left\lceil \frac{BR_i^{(0)}}{T_j} \right\rceil - 1 \right) BC_j = BC_i + \sum_{j<i} \left(\left\lceil \frac{WR_i}{T_j} \right\rceil - 1 \right) BC_j \\ &\leq WC_i + \sum_{j<i} \left\lceil \frac{WR_i}{T_j} \right\rceil WC_j = WR_i = BR_i^{(0)}. \end{aligned}$$

Next, if $BR_i^{(l+1)} \leq BR_i^{(l)}$, then we can conclude from (5.4) that also $BR_i^{(l+2)} \leq BR_i^{(l+1)}$, as filling in a lower value in the right-hand side of (5.4) gives a lower or equal result.

Next, we prove that the sequence can only take on a finite number of values. To this end, we note that $BR_i^{(l)}$ is bounded from below by BC_i and from above by WR_i. This means that each factor

$$\left\lceil \frac{BR_i^{(l)}}{T_j} \right\rceil - 1$$

in (5.4) can only take on a finite number of values. Combining this for all higher priority tasks τ_j, we can conclude that the right-hand side of (5.4) can only take on a finite number of values.

We finally prove $BR_i^{(l)} \geq BR_i$, for all $l = 0, 1, \ldots$, by induction. Obviously, $BR_i^{(0)} = WR_i \geq BR_i$. Next, if $BR_i^{(l)}$ is an upper bound on BR_i, then

$$\sum_{j<i} \left(\left\lceil \frac{BR_i^{(l)}}{T_j} \right\rceil - 1 \right) BC_j$$

is an upper bound on the amount of preemption of the best-case execution of τ_i by higher priority tasks, and hence $BR_i^{(l+1)}$ is also an upper bound on BR_i. □

In order to get a better initialization value in (5.3), we may replace the worst-case response times based on *worst-case computation* times, by worst-case response times based on *best-case computation* times. Without further elaboration we mention that for $i > 1$, $WR_i - WR_{i-1}$ is also an upper bound on the best-case response time BR_i, so we can use this value too for initialization.

5.2.4 An example

To illustrate the iterative procedure for best-case response times, consider the example of Figure 3.3, of which characteristics are given in Table 3.1. Figure 5.4 shows the successive iterations of the computation of the best-case response time of task τ_3, yielding an eventual value of 22. As we can see in this figure, the initial upper bound $BR_3^{(0)} = 56$, indicated by a dashed line, falls inside the period of execution -6 of τ_1 and the period of execution -3 of τ_2. As a result, we can conclude that these executions do not preempt a best-case execution of τ_3, so the preemptions can at most contain executions $-5,\ldots,-1$ of τ_1 and executions -2 and -1 of τ_2, resulting in a new upper bound of $BR_3^{(1)} = 5+5\cdot 3+2\cdot 11 = 42$.

Next, in iteration (ii), we see that execution -5 of τ_1 does not preempt the best-case execution of τ_3 either, so the preemptions can at most contain executions $-4,\ldots,-1$ of τ_1 and executions -2 and -1 of τ_2, resulting in a new upper bound of $BR_3^{(2)} = 5+4\cdot 3+2\cdot 11 = 39$.

This continues up to iteration (vi). There, we see that executions -2 and -1 of τ_1 and execution -1 of τ_2 preempt the best-case execution of τ_3, resulting in $BR_3^{(6)} = 5+2\cdot 3+1\cdot 11 = 22$. As $BR_3^{(6)} = BR_3^{(7)}$, the procedure stops. The best-case response time is therefore 22, of which the execution and preemptions are shown in Figure 5.3.

Using the techniques described by Palencia Gutiérrez, Gutiérrez García & González Harbour [1998] and Kim, Lee, Shin & Chang [2000] results in the same value for the best-case response times of τ_1 and τ_2, but a lower bound of only 5 for task τ_3, which is quite far off the actual best-case response time of 22. The difference between these results may be explained as follows. First, a best-case execution of a task contains direct preemptions by higher priority tasks. Secondly, some executions of higher priority tasks may themselves be preempted by even-higher priority tasks, moving the former ones into the interval of the best-case execution of the considered task. This happens in the above described example. In our approach, where we start with an upper bound on the best-case response time, that is gradually reduced, these 'preemption-caused preemptions' are included, whereas in the approaches by Palencia Gutiérrez et al. [1998] and Kim et al. [2000] they may be missed. The techniques described by Redell & Sanfridson [2002] and Redell [2003] are similar to ours.

5.3 Occupied times

The best-case occupied time is the dual of the worst-case occupied time. The *best-case occupied time* (*BO*) of a task τ is the shortest possible span of time from a release of τ till the moment in time that τ can start or resume its execution after

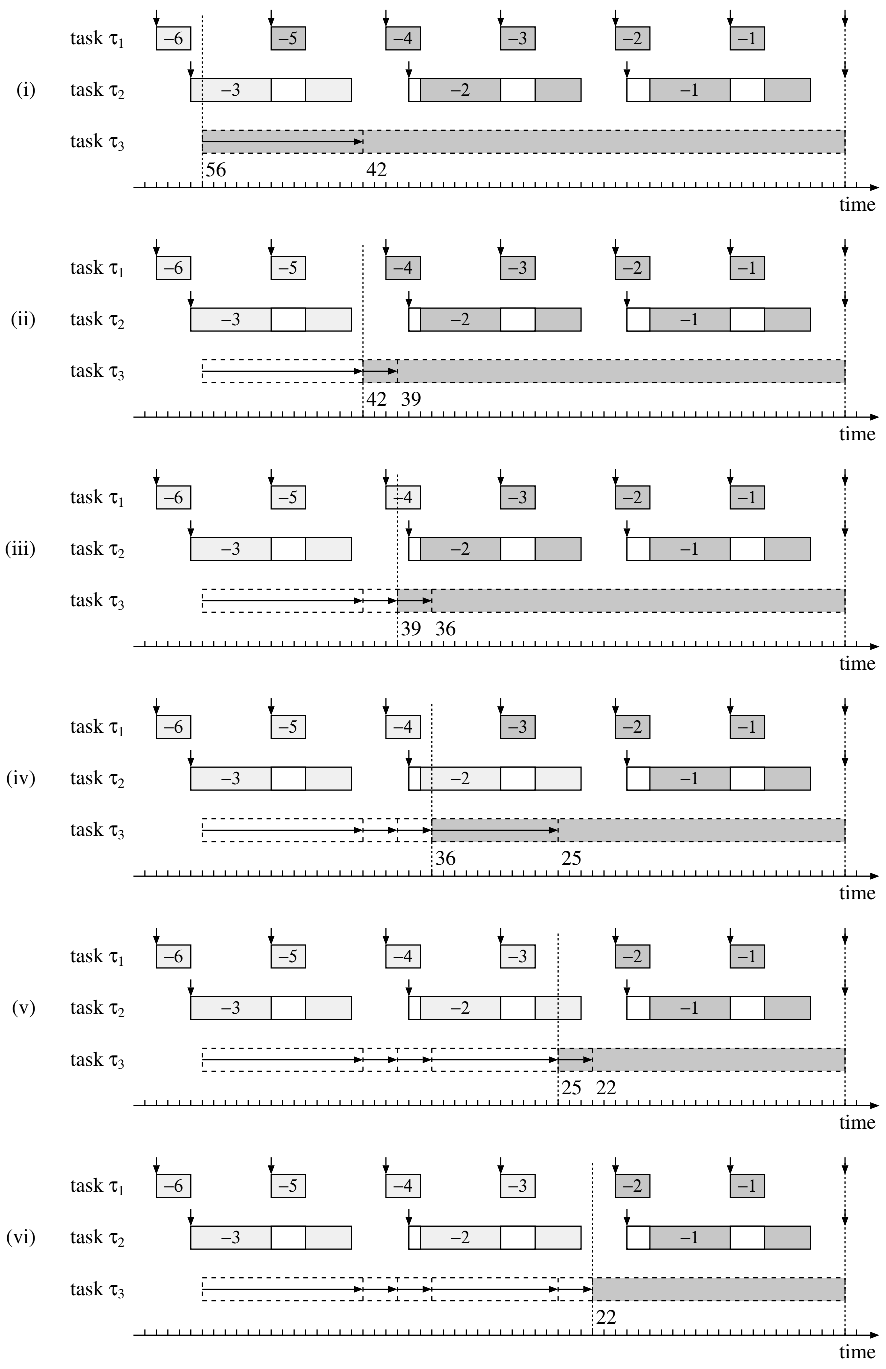


Figure 5.4. Iterations to determine the best-case response time of task τ_3. Executions are again numbered for ease of reference.

completion of a computation C.

Figure 5.5 illustrates the best-case response time BR_3 and best-case occupied time BO_3 of task τ_3 as functions of the computation time C_3. Similarly to the worst-case notions, the difference between both best-case notions is in the (open or closed) end-points of the line fragments. Further note that BR_3 is only defined for positive computation times, whereas BO_3 is also defined for a computation time of zero.

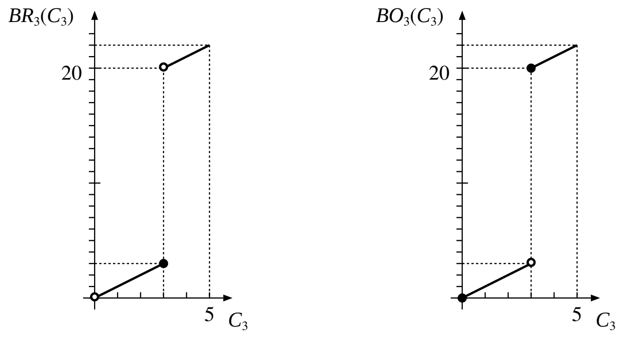


Figure 5.5. The best-case response time BR_3 and best-case occupied time BO_3 of task τ_3 as functions of the computation time C_3.

For a computation time $C = 0$, the best-case occupied time is equal to the best-case start time, i.e.

$$BS_i = BO_i(0). \tag{5.5}$$

In the remainder of this document, we will therefore treat the best-case start time as a special case of the best-case occupied time. Given Lemma 5.1, the best-case start time of a task is equal to zero.

We will first determine the best-case occupied time of a task τ by construction using a time line with an optimal instant for τ. A next theorem presents a recursive equation for the best-case occupied time of a task. That equation is derived from the recursive equation (5.2) for the best-case response time in a similar way as the worst-case occupied time has been derived from the recursive equation (4.2) for the worst-case response time. Similarly to the presentation of worst-case occupied times in Section 4.3, we therefore revisit best-case response times before presenting the theorem.

5.3.1 A time line

The best-case response time was constructed in Section 5.2.1 by drawing the response of the considered task *backwards* in time. The best-case occupied time is constructed similarly. Considering Figure 5.3, the response of task τ_3 is preempted by higher priority tasks for a duration of 17 *before* τ_3 completed its remaining computation time of 3. Hence, the best-case occupied time $BO_3(3) = 20 > 3 = BR_3(3)$. For a computation time $C_3 = 2$, task τ_3 can extend its best-case execution without preemptions. The best-case occupied time and the best-case response time are therefore the same for a computation time $C_3 = 2$, i.e. $BO_3(2) = BR_3(2) = 2$.

From the construction of the best-case occupied time by means of a time line, we draw the following conclusion.

Corollary 5.3. *Let task τ have minimal response time BR given a release at time a and a completion at the simultaneous release of all task with a higher priority than τ. In such a case, the best-case occupied time BO is identical to BR if and only if no higher priority task completed its execution at time a.* □

5.3.2 Response time revisited

From Lemma 4.4 and the proof of Theorem 5.2, we draw the following conclusion.

Corollary 5.4. *There exists a positive solution for the recursive equation (5.2) for the best-case response time BR_i of a task τ_i if $U_{i-1} < 1$.* □

Moreover, we draw the following conclusion from the construction of the best-case response time by means of a time line, similar to Corollary 4.3.

Corollary 5.5. *For $U_{i-1} < 1$ and $C_i > 0$, the smallest positive solution $BR_i(C_i)$ of (5.2) has the following properties.*

(i) $BR_i(C_i)$ is a defined and strictly increasing function of C_i.

(ii) $BR_i(C_i)$ is discontinuous at points C_i where $BR_i(C_i) = kT_j$, with $k \in \mathbb{Z}^+$ and $j < i$.

(iii) $BR_i(C_i)$ is left-continuous, i.e.

$$\lim_{x \uparrow C_i} BR_i(x) = BR_i(C_i). \tag{5.6}$$

□

The *best-case occupied time $BO_i(C_i)$* of task τ_i with computation time $C_i \geq 0$ is given by

$$BO_i(C_i) = \lim_{x \downarrow C_i} BR_i(x). \tag{5.7}$$

5.3.3 A recursive equation

Similarly to the derivation of the worst-case occupied time from the worst-case response time, we will use (5.7) as a starting point for the derivation of the recursive equation for the best-case occupied time from the best-case response time, and use Lemma 4.5.

Theorem 5.4. *When the smallest positive solution of (4.2) for a computation time BC'_i is at most $\min(D_i, T_i)$, the best-case occupied time BO_i of τ_i with a computation time $BC_i \in [0, BC'_i]$ is given by the largest non-negative $x \in \mathbb{R}$ that satisfies*

$$x = BC_i + \sum_{j<i} \left\lfloor \frac{x}{T_j} \right\rfloor BC_j. \tag{5.8}$$

Proof. Given Lemma 4.5, we can make the following derivation starting from (5.7).

$$\begin{aligned}
BO_i(BC_i) &= \lim_{x \downarrow BC_i} BR_i(x) \\
&= \{(5.2)\} \lim_{x \downarrow BC_i} \left(x + \sum_{j<i} \left(\left\lceil \frac{BR_i(x)}{T_j} \right\rceil - 1 \right) BC_j \right) \\
&= BC_i + \sum_{j<i} \lim_{x \downarrow BC_i} \left(\left\lceil \frac{BR_i(x)}{T_j} \right\rceil - 1 \right) BC_j \\
&= \{\text{Lemma 4.5}\} BC_i + \sum_{j<i} \left\lfloor \lim_{x \downarrow BC_i} \frac{BR_i(x)}{T_j} \right\rfloor BC_j \\
&= \{(5.7)\} BC_i + \sum_{j<i} \left\lfloor \frac{BO_i(BC_i)}{T_j} \right\rfloor BC_j
\end{aligned}$$

Remember that the largest positive solution of (5.2) only considers a single job of the task under consideration. The same holds for the largest non-negative solution of (5.8). Hence, the best-case occupied time BO_i of task τ_i with a computation time $BC_i \in [0, BC'_i]$ is the largest non-negative x satisfying (5.8), when $WR_i(BC'_i) \leq \min(D_i, T_i)$. □

5.3.4 An iterative procedure

The next theorem describes how the largest positive solution of (5.4) can be found using an iterative procedure.

First, we prove the following lemma providing an upper bound for the best-case occupied time BO_i of task τ_i.

Lemma 5.3. *The value $\iota_i^{\mathrm{AB}}(BC_i)$ defined by*

$$\iota_i^{\mathrm{AB}}(BC_i) = \frac{BC_i}{(1 - BU_{i-1})}, \tag{5.9}$$

is an appropriate initial value for the iterative procedure to determine $BO_i(BC_i)$.

Proof. The best-case occupied time $BO_i(BC_i)$ is the largest value of x satisfying (5.8), and the iterative procedure therefore has to start with an upper bound. Hence, we have to prove that $\iota_i^{\text{AB}}(BC_i)$ is an upper bound for $BO_i(BC_i)$. To this end, we derive

$$\begin{aligned} BO_i(BC_i) &= BC_i + \sum_{j<i} \left\lfloor \frac{BO_i(BC_i)}{T_j} \right\rfloor BC_j \\ &\leq BC_i + \sum_{j<i} \left(\frac{BO_i(BC_i)}{T_j} \right) BC_j = C_i + BO_i(BC_i)BU_{i-1}. \end{aligned}$$

Hence for $BU_{i-1} < 1$, we get $BO_i(BC_i) \leq BC_i/(1 - BU_{i-1})$. □

Theorem 5.5. *When the smallest positive solution of (4.2) is at most min(D_i, T_i), the best-case occupied time BO_i of task τ_i can be found by the following iterative procedure.*

$$BO_i^{(0)} = \iota_i^{\text{AB}}(BC_i) \qquad (5.10)$$

$$BO_i^{(l+1)} = BC_i + \sum_{j<i} \left\lfloor \frac{BO_i^{(l)}}{T_j} \right\rfloor BC_j, \quad l = 0, 1, \ldots \qquad (5.11)$$

The procedure is stopped when the same value is found for two successive iterations of l.

Proof. The proof is similar to the proof of Theorem 5.3. □

5.4 Discussion

In this section, we consider best-case execution times, initial values for the iterative procedure to determine the best-case occupied time, present several properties of the best-case occupied time, introduce the notion of *best-case induced load*, and finally provide an overview of the equations for response times and occupied times, and the induced load.

5.4.1 Execution times

Given (3.4) and the definition of the execution interval $e_{ik} = r_{ik} - s_{ik}$, we immediately derive the following conclusion.

Corollary 5.6. *The best-case execution time BE_i of a task τ_i is equal to the best-case response time BR_i of that task.* □

5.4.2 Initial values

In this section, we consider initial values for the iterative procedures to determine the best-case response time and the best-case occupied time.

Standard initial value

In Theorem 5.3, we used $WR_i(WC_i)$ as the initial value for the iterative procedure to calculate BR_i. As mentioned before, $WR_i - WR_{i-1}$ is also an upper bound on the best-case response time BR_i, and therefore an attractive alternative when WR_{i-1} is known. In both cases, the initial value is WR_i minus a constant. We will therefore use the term $\iota_i^{\mathrm{SB}}(WC_i)$ to denote a standard initial value, which is defined using a constant χ_i^{B}.

$$\iota_i^{\mathrm{SB}}(WC_i) = WR_i(WC_i) - \chi_i^{\mathrm{B}} \tag{5.12}$$

In this document, we assume $\chi_i^{\mathrm{B}} = WR_{i-1}$ for the iterative procedure to calculate BR_i. Unfortunately, $WR_i - WR_{i-1}$ is not an upper bound on the best-case occupied time BO_i in general. We therefore assume $\chi_i^{\mathrm{B}} = 0$ for the iterative procedure to calculate BO_i.

Alternative initial value

The alternative initial value $\iota_i^{\mathrm{AB}}(BC_i)$, which is defined in Lemma 5.3, is an appropriate initial value for the iterative procedures to calculate both BR_i and BO_i. From Lemma 5.3 and Lemma 4.8, which defines the alternative initial values $\iota_i^{\mathrm{AW}}(WC_i)$, we draw the following conclusion.

Corollary 5.7. *The best-case occupied time BO_i of a task τ_i is less than or equal to the worst-case response time WR_i of that task, i.e.*

$$BO_i \leq WR_i. \tag{5.13}$$

□

Moreover, in combination with (4.7), (5.7), and $BC_j \leq WC_j$ for $j \leq i$ we draw the following conclusion.

Corollary 5.8. *The following relation holds between response times and occupied times of a task τ_i.*

$$BR_i(BC_i) \leq BO_i(BC_i) \leq \frac{BC_i}{1 - BU_{i-1}} \leq \frac{WC_i}{1 - WU_{i-1}} \leq WR_i(WC_i) \leq WO_i(WC_i) \tag{5.14}$$

□

As mentioned before, the worst-case and best-case computation times are chosen to be equal ($WC_i = BC_i$) in most examples we use, and we simply use the computation time C_i. For ease of presentation, we therefore introduce an alternative initial value $\iota_i^{\mathrm{A}}(C_i)$, which is defined as

$$\iota_i^{\mathrm{A}}(C_i) = \frac{C_i}{(1 - U_{i-1})}. \tag{5.15}$$

Combined initial value

Similar to worst-case analysis, we propose an initial value for best-case analysis that combines standard and alternative initial values. This combined initial value $\iota_i^{\mathrm{CB}}(BC_i)$ is defined as

$$\iota_i^{\mathrm{CB}}(BC_i) = \min(\iota_i^{\mathrm{SB}}(WC_i), \iota_i^{\mathrm{AB}}(BC_i)). \tag{5.16}$$

5.4.3 Properties of the best-case occupied time

The following two lemmas are the best-case equivalents of Lemma 4.9 and Lemma 4.10 from the worst-case analysis.

Lemma 5.4. *The best-case response time BR_i of task τ_i is not a multiple of any period T_j for $j < i$.*

Proof. Similar to the proof of Lemma 4.9. □

Lemma 5.5. *The best-case response time BR_i of τ_i is a solution of the recursive equation for the best-case occupied time (5.8).*

Proof. Similar to the proof of Lemma 4.10 using Lemma 5.4. □

From the time line as graphical representation of (5.8) we draw the following conclusion.

Corollary 5.9. *For $U_{i-1} < 1$ and $C_i \geq 0$, the smallest non-negative solution $BO_i(C_i)$ of (5.8) has the following properties:*
(i) $BO_i(C_i)$ is a defined and strictly increasing function of C_i;
(ii) $BO_i(C_i)$ is discontinuous at points C_i where $BR_i(C_i)$ is discontinuous;
(iii) $BO_i(C_i)$ is right-continuous, i.e.

$$\lim_{x \downarrow C_i} BO_i(x) = BO_i(C_i). \tag{5.17}$$

□

5.4.4 Induced load

The summation term in the recursive equation (5.2) for the best-case response time is also termed the *best-case induced load*

$$BL_{i-1}(x) = \sum_{j<i} \left(\left\lceil \frac{x}{T_j} \right\rceil - 1 \right) BC_j, \tag{5.18}$$

of tasks τ_1 till τ_{i-1} in an open interval of length x, e.g. $(t, t+x)$. The best-case induced load is the *minimal* load induced by tasks with a higher priority than task τ_i.

Similarly, the summation term in (5.8) for the best-case occupied time is the best-case induced load

$$BL^{+}_{i-1}(x) = \sum_{j<i} \left\lfloor \frac{x}{T_j} \right\rfloor BC_j. \tag{5.19}$$

of tasks τ_1 till τ_{i-1} in a right-open interval of length x, e.g. $[t, t+x)$.

5.4.5 Overview of equations

Together with the previous chapter, this chapter presented a basic set of recursive equations for best-case analysis and worst-case analysis. An overview of this set is given in Table 5.1. Note that the recursive equations for best-case analysis closely resemble those for worst-case analysis, apart from an additional term -1 for the response times and a lacking term $+1$ for the occupied times. For best-case analysis, the largest values satisfying the recursive equations yield the times we are looking for, and the associated iterative procedures therefore start with an upper bound. For worst-case analysis, we need the smallest values satisfying the recursive equations, and the iterative procedures therefore start with a lower bound.

Table 5.1. Overview of recursive equations for response times and occupied times.

	response times	occupied times
best-case	$x = BC_i + \sum_{j<i} \left(\left\lceil \frac{x}{T_j} \right\rceil - 1 \right) BC_j$	$x = BC_i + \sum_{j<i} \left\lfloor \frac{x}{T_j} \right\rfloor BC_j$
worst-case	$x = WC_i + \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j$	$x = WC_i + \sum_{j<i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor + 1 \right) WC_j$

An overview of the equations for the best-case and worst-case induced load is given in Table 5.2. Note that $WL_i(x)$ is the *maximal* and $BL^{+}_i(x)$ is the *minimal* induced load in a right-open interval of length x, e.g. $[t, t+x)$.

Table 5.2. Overview of equations for exclusive induced load and inclusive induced load.

	induced load (exclusive)	induced load (inclusive)
best-case	$BL_i(x) = \sum_{j<i} \left(\left\lceil \frac{x}{T_j} \right\rceil - 1 \right) BC_j$	$BL^{+}_i(x) = \sum_{j<i} \left\lfloor \frac{x}{T_j} \right\rfloor BC_j$
worst-case	$WL_i(x) = \sum_{j<i} \left\lceil \frac{x}{T_j} \right\rceil WC_j$	$WL^{+}_i(x) = \sum_{j<i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor + 1 \right) WC_j$

6

Calculating Response Times and Occupied Times

As discussed before, many real-time systems needing an online schedulability test require exact schedulability analysis. For this analysis, we typically need to calculate response times, and sometimes occupied times as well (see Chapter 8). This chapter therefore evaluates initial values for the iterative procedures to calculate response times and occupied times of periodic tasks under FPPS and arbitrary phasing. We use a model based on discrete scheduling, i.e. all task parameters are integers and preemptions are restricted to integer time values.

We show that the number of iterations needed to determine the worst-case response time of a task using a standard initial value, such as (4.12), increases logarithmically for an increasing worst-case computation time of that task. We prove that the number of iterations for the alternative initial value presented in Lemma 4.8 is periodic and bounded by a constant. The costs in terms of required execution time to determine worst-case response times using a standard and the combined initial value (4.14) are compared by means of an experiment.

In this chapter, we show that the number of iterations needed to determine the best-case response time and the best-case occupied time of a task is periodic and bounded by a constant for the alternative initial value presented in Lemma 5.3 as well as for standard initial values. The case for worst-case occupied times is similar to the case for worst-case response times.

6.1 Discrete scheduling

As described in Section 3.4.2, discrete scheduling is based on a model in which all task parameters are integers, i.e. $T_i, C_i, D_i \in \mathbb{Z}^+$ and $\varphi_i \in \mathbb{Z}$ for $1 \leq i \leq n$, and preemptions are restricted to integer time values. One of the advantages of discrete scheduling is that it allows the usage of the notion of *hyperperiod*. We will describe the notion of hyperperiod and present specific properties of response times and occupied times related with that notion. We start with a presentation of dedicated equations for discrete scheduling describing occupied times in terms of response times.

6.1.1 Response times and occupied times

For continuous scheduling, the worst-case occupied time WO_i of a task τ_i is described in terms of the worst-case response time WR_i using a limit; see (4.7). For discrete scheduling, this description can be simplified as follows.

Theorem 6.1. *For discrete scheduling, we can determine the worst-case occupied time of a task τ_i by means of*

$$WO_i(C_i) = WR_i(C_i + 1) - 1, \tag{6.1}$$

given that $WR_i(C_i) \leq \min(D_i, T_i)$.

Proof. We consider two complementary cases in the proof, i.e. whether or not the worst-case response time of task τ_i equals a multiple of the period of a task with a higher priority than τ_i.

Let $WR_i(C_i) = mT_j$ for some $m \in \mathbb{N}^+$ and $j < i$. In such a case, task τ_j is released at $WR_i(C_i)$, and therefore $WO_i(C_i) \neq WR_i(C_i)$. Because all task parameters are integers and preemptions are restricted to integer time values for discrete scheduling, task τ_i can execute an additional unit of time non-preemptively when it can resume its execution at $WO_i(C_i)$, hence, $WO_i(C_i) = WR_i(C_i + 1) - 1$.

Let $WR_i(C_i) \neq mT_j$ for all $m \in \mathbb{N}^+$ and $j < i$. In such a case, task τ_i can continue its execution at time $WR_i(C_i)$, hence, $WO_i(C_i) = WR_i(C_i)$. Similar to the previous case, τ_i can execute an additional unit of time non-preemptively when it completes the execution of C_i at $WR_i(C_i)$, hence, $WO_i(C_i) = WR_i(C_i + 1) - 1$. □

Similarly to Theorem 6.1 with an equation for the worst-case occupied time for discrete scheduling, we have the following theorem with an equation for the best-case occupied time for discrete scheduling.

Theorem 6.2. *For discrete scheduling, we can determine the best-case occupied time of a task* τ_i *by means of*

$$BO_i(BC_i) = BR_i(C_i + 1) - 1, \tag{6.2}$$

given that $WR_i(C_i) \leq \min(D_i, T_i)$.

Proof. Similar to the proof of Theorem 6.1. □

6.1.2 Hyperperiod

The hyperperiod H_{i-1} of the tasks τ_1 till τ_{i-1} is an interval corresponding to the least common multiple of the periods of these tasks, i.e. $H_{i-1} = \mathrm{lcm}(T_1, \ldots, T_{i-1})$. Hence, H_{i-1} is an integer multiple of the periods of these tasks, i.e. for $1 \leq j \leq i-1$ the hyperperiod H_{i-1} satisfies

$$\left\lfloor \frac{H_{i-1}}{T_j} \right\rfloor = \left\lceil \frac{H_{i-1}}{T_j} \right\rceil = \frac{H_{i-1}}{T_j}. \tag{6.3}$$

The activation pattern of the tasks τ_1 till τ_{i-1} recurs after the hyperperiod H_{i-1}, and a level-$i-1$ schedule σ_{i-1} repeats itself every H_{i-1} units of time [Leung & Merrill, 1980]. Stated in other words, a level-$i-1$ schedule σ_{i-1} is periodic with period H_{i-1}.

The amount of slack time S_{i-1} produced by (i.e. the time that is not used for executions of) tasks τ_1 till τ_{i-1} in the hyperperiod H_{i-1} is given by

$$\begin{aligned} S_{i-1} &= H_{i-1} - \sum_{j<i} \frac{H_{i-1}}{T_j} C_j \\ &= H_{i-1}(1 - U_{i-1}). \end{aligned} \tag{6.4}$$

When the computation time C_i of a task τ_i is a positive integral multiple of this slack S_{i-1}, the worst-case response time WR_i of τ_i also equals the same multiple of the hyperperiod H_{i-1}.

Lemma 6.1. *For arbitrary* $k \in \mathbb{N}^+$ *for which* $T_i \geq kH_{i-1}$, *the following equation holds.*

$$WR_i(kS_{i-1}) = kH_{i-1} \tag{6.5}$$

Proof. The lemma is proven by using $\iota_i^{\mathrm{A}}(kS_{i-1})$ (5.15) as initial value to calculate $WR_i(kS_{i-1})$.

$$\begin{aligned} WR_i^{(0)}(kS_{i-1}) &= \{\text{Lemma 4.8 and (5.15)}\} \; \frac{kS_{i-1}}{(1-U_{i-1})} \\ &= \{(6.4)\} \; kH_{i-1} \end{aligned}$$

$$\begin{aligned}
WR_i^{(1)}(kS_{i-1}) &= \{(4.4)\}\ kS_{i-1} + \sum_{j<i} \left\lceil \frac{WR_i^{(0)}(kS_{i-1})}{T_j} \right\rceil C_j \\
&= \{(6.4)\}\ kH_{i-1}(1-U_{i-1}) + \sum_{j<i} \left\lceil \frac{kH_{i-1}}{T_j} \right\rceil C_j \\
&= \{(6.3)\}\ kH_{i-1}(1-U_{i-1}) + kH_{i-1} \sum_{j<i} \frac{C_j}{T_j} \\
&= \{(3.8) \text{ and } (3.9)\}\ kH_{i-1}(1-U_{i-1}) + kH_{i-1}U_{i-1} \\
&= kH_{i-1}
\end{aligned}$$

Hence, the iterative procedure stops directly with $WR_i(kS_{i-1}) = kH_{i-1}$. □

For the best-case occupied time, a similar lemma can be stated.

Lemma 6.2. *For arbitrary $k \in \mathbb{N}^+$ for which $T_i \geq kH_{i-1}$, the following equation holds.*

$$BO_i(kS_{i-1}) = kH_{i-1} \tag{6.6}$$

Proof. The proof of this lemma is similar to that of Lemma 6.1 using $\iota_i^{\mathrm{A}}(kS_{i-1})$ as an initial value to calculate $BO_i(kS_{i-1})$. □

The following lemma expresses the best-case occupied time in terms of the worst-case response time, and the best-case response time in terms of the worst-case occupied time. The proof of the lemma is based on the construction of the response times (see Sections 4.2.1 and 5.2.1) and occupied times (see Sections 4.3.1 and 5.3.1).

Lemma 6.3. *For arbitrary $k \in \mathbb{N}^+$ and $T_i \geq kH_{i-1}$, the following equations hold.*

$$BO_i(C_i) = kH_{i-1} - WR_i(kS_{i-1} - C_i) \ \ \textit{for}\ 0 \leq C_i < kS_{i-1} \tag{6.7}$$

$$BR_i(C_i) = kH_{i-1} - WO_i(kS_{i-1} - C_i) \ \ \textit{for}\ 0 < C_i \leq kS_{i-1} \tag{6.8}$$

Proof. Below, we will prove (6.8). The proof of (6.7) is similar.

Let all tasks τ_j with $j < i$ be released at time zero. The amount of slack in the interval $[0, kH_{i-1}]$ is kS_{i-1}, as (6.5) indicates. For an appropriate phasing φ_i and computation time C_i for task τ_i, time kH_{i-1} becomes an *optimal instant* for τ_i. Let $0 < C_i \leq kS_{i-1}$. The best-case response time $BR_i(C_i)$ for C_i of τ_i may be found by going *backwards* in time from time kH_{i-1}, using C_i time units of the slack kS_{i-1}. The remaining slack in the interval $[0, kH_{i-1}]$ equals $kS_{i-1} - C_i$. For an appropriate phasing φ_i and computation time $kS_{i-1} - C_i$ for task τ_i, time zero becomes a *critical instant* of τ_i. Whenever task τ_i would execute an additional amount $\varepsilon > 0$ from time $WR_i(kS_{i-1} - C_i)$, this would therefore overlap with the start of the best-case execution that starts at $kH_{i-1} - BR_i(C_i)$. Therefore, time $kH_{i-1} - BR_i(C_i)$ is equal to the worst-case occupied time $WO_i(kS_{i-1} - C_i)$, resulting in (6.8). □

We mention that (6.7) can be derived from (6.8) using (6.1) and (6.2), and vice versa.

6.2 Numbers of iterations

In this section, we will consider the number of iterations $\omega_i(C_i, \iota_i(C_i))$ needed to calculate $WR_i(C_i)$ as a function of C_i for both the standard initial value $\iota_i^{\mathrm{SW}}(C_i) = \chi_i^{\mathrm{W}} + C_i$ (4.12) and the alternative initial value $\iota_i^{\mathrm{A}}(C_i) = C_i/(1 - U_{i-1})$ (5.15). We will use $\omega_i^{\mathrm{S}}(C_i)$ and $\omega_i^{\mathrm{A}}(C_i)$ as shorthands for $\omega_i(C_i, \iota_i^{\mathrm{SW}}(C_i))$ and $\omega_i(C_i, \iota_i^{\mathrm{A}}(C_i))$, respectively. For illustration purposes, we use the example of Table 3.1, and use the computation time C_3, period T_3, and deadline D_3 of task τ_3 as variables. We assume $T_3 = D_3$, $T_3 > T_2$, and D_3 sufficiently large such that the worst-case response time $WR_3(C_3) \leq D_3$ for all C_3 we use. For our example, we show that $\omega_3^{\mathrm{S}}(C_3)$ has the shape of a logarithmic function, whereas $\omega_3^{\mathrm{A}}(C_3)$ is bounded by a constant (i.e. 5) and periodic. In Section 6.2.2, we prove that $\omega_i^{\mathrm{A}}(C_i)$ is bounded by a constant and periodic.

6.2.1 Standard initial value

Using (6.4), we get for the example of Table 3.1 a slack $S_2 = H_2(1 - U_2) = 190(1 - (3/10 + 11/19)) = 23$. Figure 6.1 shows the number of iterations $\omega_3^{\mathrm{S}}(C_3)$ needed to calculate $WR_3(C_3)$ as a function of C_3 for $C_3 = 1, \ldots, 2300$ (i.e. 100 times the slack S_2). Note that the graph has the shape of a logarithmic function, but is not strictly non-decreasing in C_3.

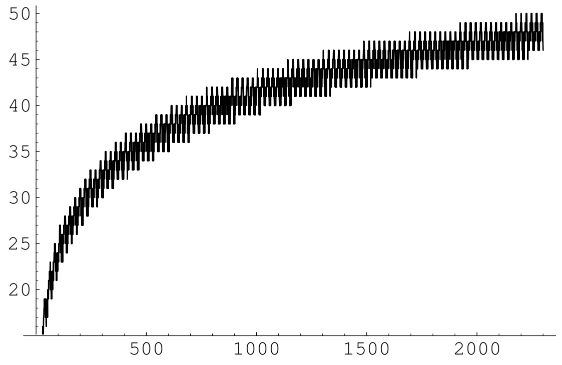


Figure 6.1. The number of iterations $\omega_3^{\mathrm{S}}(C_3)$ needed to calculate $WR_3(C_3)$ as a function of C_3 for $C_3 = 1, \ldots, 2300$.

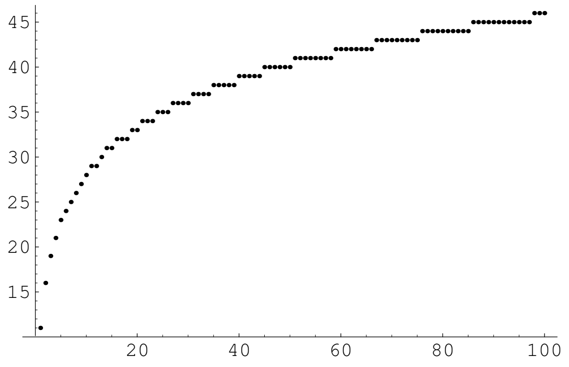


Figure 6.2. The number of iterations $\omega_3^{\mathrm{S}}(kS_2)$ needed to calculate $WR_3(kS_2)$ using the standard initial value ι_3^{SW} for $k = 1, \ldots, 100$.

Figure 6.2 shows $\omega_3^{\mathrm{S}}(kS_2)$ for $k = 1, \ldots, 100$, i.e. the same function, but with parameters restricted to positive integral multiples of the slack S_2.

For these restricted values of C_3, $\omega_3^{\mathrm{S}}(C_3)$ is strictly non-decreasing, and the resulting graph closely follows a logarithmic function. Below, we will derive that $\omega_i^{\mathrm{S}}(kS_{i-1})$, where $k \in \mathbb{N}^+$, can indeed be approximated by a logarithmic function $\omega_i^l(kS_{i-1})$ that is defined as

$$\omega_i^l(kS_{i-1}) = {}^{U_{i-1}}\!\log \frac{\Delta_{i-1}U_{i-1}}{kS_{i-1}U_{i-1} + \Delta_{i-1} - \chi_i^{\mathrm{W}}(1 - U_{i-1})}. \tag{6.9}$$

Note that $\omega_i^l(kS_{i-1})$ is logarithmic in kS_{i-1}, rather than $1/(kS_{i-1})$, because $U_{i-1} < 1$. The term Δ_{i-1} in this equation is a constant, a so-called *cumulative average rounding-off value*. For the derivation of (6.9), it is assumed that Δ_{i-1} can be used to approximate the value that is added to each iteration $WR_i^{(l)}(C_i)$ due to the application of the ceiling function, i.e.

$$\sum_{j<i} \left\lceil \frac{WR_i^{(l)}(C_i)}{T_j} \right\rceil C_j \approx WR_i^{(l)}(C_i)U_{i-1} + \Delta_{i-1}. \tag{6.10}$$

In order to arrive at a value for Δ_{i-1}, we assumed that we can use the average of the fractions 0, $1/T_j$ till $(T_j - 1)/T_j$ as average rounding-off value for the term $WR_i^{(l)}(C_i)/T_j$ for each $j < i$. Hence it is assumed that the *average rounding-off value* Δ_j^{τ} per iteration added to $WR_i^{(l)}$ due to rounding off in $\left\lceil WR_i^{(l)}(C_i)/T_j \right\rceil C_j$ is given by

$$\Delta_j^{\tau} = \frac{C_j(T_j - 1)}{2T_j} = U_j \frac{(T_j - 1)}{2}.$$

Adding this up for tasks τ_1 till τ_{i-1}, Δ_{i-1} is then given by

$$\Delta_{i-1} = \sum_{j<i} \Delta_j^\tau = \sum_{j<i} U_j^\tau \frac{(T_j - 1)}{2}. \tag{6.11}$$

Given the assumption above, we first show by induction that

$$WR_i^{(l)}(C_i) \approx (C_i + \Delta_{i-1}) \sum_{k=0}^{l} U_{i-1}^k + (\chi_i^{\mathrm{W}} - \Delta_{i-1}) U_{i-1}^l \tag{6.12}$$

holds for $l = 0, \ldots, \lambda - 1$, where λ denotes the number of iterations after which the iterative procedure stops. To this end, we assume an initial value of $\chi_i^{\mathrm{W}} + C_i$ for the first iteration, and note that $WR_i^{(0)}(C_i) = \chi_i^{\mathrm{W}} + C_i$. Hence, for $l = 0$, the approximation sign can be replaced by the equality sign. Next, for $WR_i^{(1)}(C_i)$ we get

$$\begin{aligned}
WR_i^{(1)}(C_i) &= C_i + \sum_{j<i} \left\lceil \frac{WR_i^{(0)}(C_i)}{T_j} \right\rceil C_j \\
&\approx \{(6.10)\}\ C_i + WR_i^{(0)}(C_i) U_{i-1} + \Delta_{i-1} \\
&= C_i + (\chi_i^{\mathrm{W}} + C_i) U_{i-1} + \Delta_{i-1} \\
&= (C_i + \Delta_{i-1})(1 + U_{i-1}) + (\chi_i^{\mathrm{W}} - \Delta_{i-1}) U_{i-1} \\
&= (C_i + \Delta_{i-1}) \sum_{k=0}^{1} U_{i-1}^k + (\chi_i^{\mathrm{W}} - \Delta_{i-1}) U_{i-1}.
\end{aligned}$$

Moreover, if $WR_i^{(m-1)}(C_i)$ satisfies (6.12) (induction hypothesis), we get

$$\begin{aligned}
WR_i^{(m)}(C_i) &= C_i + \sum_{j<i} \left\lceil \frac{WR_i^{(m-1)}(C_i)}{T_j} \right\rceil C_j \\
&\approx \{(6.10)\}\ C_i + WR_i^{(m-1)}(C_i) U_{i-1} + \Delta_{i-1} \\
&\approx \{\text{induction hypothesis}\} \\
&\quad C_i + \left((C_i + \Delta_{i-1}) \sum_{k=0}^{m-1} U_{i-1}^k + (\chi_i^{\mathrm{W}} - \Delta_{i-1}) U_{i-1}^{m-1} \right) U_{i-1} + \Delta_{i-1} \\
&= (C_i + \Delta_{i-1}) \sum_{k=0}^{m} U_{i-1}^k + (\chi_i^{\mathrm{W}} - \Delta_{i-1}) U_{i-1}^m.
\end{aligned}$$

Hence, under assumption (6.10), we showed the validity of (6.12).

When the iterative procedure stops after λ iterations, i.e. $WR_i^{(\lambda)}(C_i) = WR_i^{(\lambda-1)}(C_i)$. $WR_i(C_i)$ is a multiple of H_{i-1} for $C_i = kS_{i-1}$ and $k \in \mathbb{N}^+$. Hence for the last iteration the evaluation of the summation does *not* give rise to an additional Δ_{i-1} (see also (6.3)).

$$\begin{aligned}
WR_i^{(\lambda)}(kS_{i-1}) &= kS_{i-1} + \sum_{j<i} \left\lceil \frac{WR_i^{(\lambda-1)}(kS_{i-1})}{T_j} \right\rceil C_j \\
&= kS_{i-1} + WR_i^{(\lambda-1)}(kS_{i-1}) U_{i-1} \\
&= kS_{i-1} + \left((kS_{i-1} + \Delta_{i-1}) \sum_{k=0}^{\lambda-1} U_{i-1}^k + (\chi_i^{\mathrm{W}} - \Delta_{i-1}) U_{i-1}^{\lambda-1} \right) U_{i-1}
\end{aligned}$$

The left-hand side of the latter equation may be re-written to $kS_{i-1}/(1-U_{i-1})$ using (6.4) and (6.5), and the term $\sum_{k=0}^{\lambda-1} U_{i-1}^k$ in the right-hand side may be rewritten to $(1-U_{i-1}^{\lambda})/(1-U_{i-1})$. By rewriting, we derive that the number of iterations λ can be approximated by a logarithmic function.

$$\begin{aligned}
& \frac{kS_{i-1}}{(1-U_{i-1})} = kS_{i-1} + \frac{(kS_{i-1}+\Delta_{i-1})U_{i-1}(1-U_{i-1}^{\lambda})}{(1-U_{i-1})} + (\chi_i^{\mathrm{W}} - \Delta_{i-1})U_{i-1}^{\lambda} \\
\Leftrightarrow\ & kS_{i-1} = \left(kS_{i-1} + (\chi_i^{\mathrm{W}} - \Delta_{i-1})U_{i-1}^{\lambda}\right)(1-U_{i-1}) + (kS_{i-1}+\Delta_{i-1})U_{i-1}(1-U_{i-1}^{\lambda}) \\
\Leftrightarrow\ & 0 = \Delta_{i-1}U_{i-1} - kS_{i-1}U_{i-1}^{\lambda+1} - \Delta_{i-1}U_{i-1}^{\lambda} + \chi_i^{\mathrm{W}} U_{i-1}^{\lambda}(1-U_{i-1}) \\
\Leftrightarrow\ & (kS_{i-1}U_{i-1} + \Delta_{i-1} - \chi_i^{\mathrm{W}}(1-U_{i-1}))U_{i-1}^{\lambda} = \Delta_{i-1}U_{i-1} \\
\Leftrightarrow\ & U_{i-1}^{\lambda} = \frac{\Delta_{i-1}U_{i-1}}{kS_{i-1}U_{i-1} + \Delta_{i-1} - \chi_i^{\mathrm{W}}(1-U_{i-1})} \\
\Leftrightarrow\ & \lambda = {}^{U_{i-1}}\!\log \frac{\Delta_{i-1}U_{i-1}}{kS_{i-1}U_{i-1} + \Delta_{i-1} - \chi_i^{\mathrm{W}}(1-U_{i-1})}
\end{aligned}$$

By replacing λ by $\omega_i'(kS_{i-1})$, we finally arrive at (6.9).

Figure 6.3 illustrates the validity of (6.9) for our example by showing the difference $\delta_3(kS_2) = \omega_3'(kS_2) - \omega_3^{\mathrm{S}}(kS_2)$ for $k = 1, \ldots, 1000$. Note that for this particular example

$$\max_{k=1}^{1000} \delta_3(kS_2) - \min_{k=1}^{1000} \delta_3(kS_2) \approx 1.$$

The derivation of the function $\omega_i'(kS_{i-1})$ is based on an unproven assumption about Δ_{i-1}. Therefore, we only made the logarithmic shape of the function $\omega_i^{\mathrm{S}}(C_i)$ plausible.

6.2.2 Alternative initial value

We will first show that, unlike $\omega_i^{\mathrm{S}}(C_i)$, $\omega_i^{\mathrm{A}}(C_i)$ does not have the shape of a logarithmic function. We subsequently consider the number of iterations needed for our example using the alternative initial value, and present our main theorem.

Based on Lemma 6.1, we draw the following conclusion.

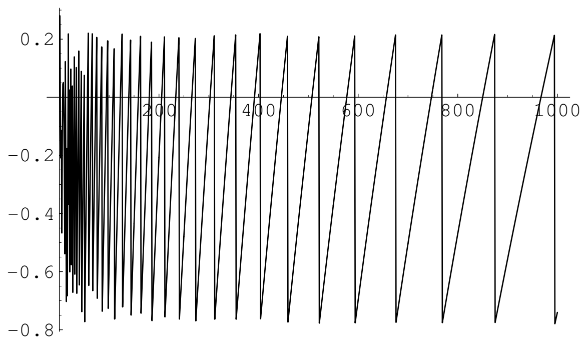


Figure 6.3. The approximated number of iterations $\omega'_3(kS_2)$ minus the actual number of iterations $\omega^S_3(kS_2)$ for $k = 1, \ldots, 1000$.

Corollary 6.1. *Only a single iteration is needed to calculate $WR_i(C_i)$ of a task τ_i when $\iota^A_i(C_i)$ is used as initial value for the iterative procedure and C_i is a multiple of the slack S_{i-1}, i.e.*

$$\omega^A_i(kS_{i-1}) = 1 \;\; \textit{for} \;\; k \in \mathbb{N}^+.$$

□

Therefore, $\omega^A_i(C_i)$ does not have the shape of a logarithmic function.

Figure 6.4 shows $\omega^A_3(C_3)$ for $C_3 = 1, \ldots, 230$ (i.e. 10 times the slack S_2). Note that $\omega^A_3(C_3)$ is bounded and periodic with period S_2.

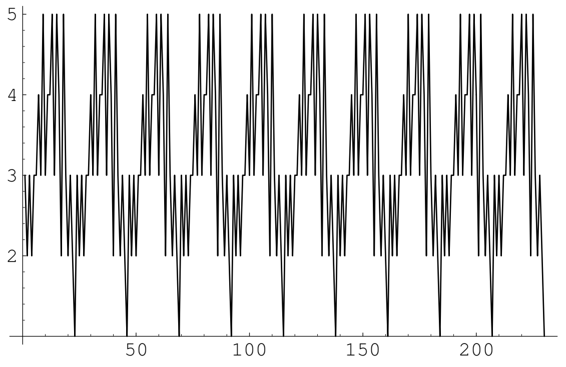


Figure 6.4. The number of iterations $\omega^A_3(C_3)$ needed to calculate $WR_3(C_3)$ using $\iota^A_3(C_3)$ as initial value for $C_3 = 1, \ldots, 230$.

Theorem 6.3 states that this behavior of $\omega^A_3(C_3)$ can be generalized to $\omega^A_i(C_i)$.

Theorem 6.3. *The function $\omega^A_i(C_i)$ is bounded by a constant and periodic with period S_{i-1}.*

Proof. For discrete scheduling and C_i in the interval $(0, S_{i-1}]$, C_i can only assume a finite number of values. For each of these values, the number of iterations needed to calculate $WR_i(C_i)$ is finite. Let M_i be the maximum number of iterations needed to calculate $WR_i(C_i)$ for C_i in the interval $(0, S_{i-1}]$. We will prove that $\omega_i^{\mathrm{A}}(C_i') = \omega_i^{\mathrm{A}}(C_i)$ for $C_i' = C_i + kS_{i-1}$ where $k \in \mathbb{N}$, i.e. that $\omega_i^{\mathrm{A}}(C_i)$ is periodic with period S_{i-1}. As a consequence, the number of iterations needed to calculate $WR_i(C_i)$ is bounded by M_i.

The proof is given in two steps. In the first step, we prove that the equation $WR_i^{(l)}(C_i') = WR_i^{(l)}(C_i) + kH_{i-1}$ holds for every iteration l by means of induction on the number of iterations. In the second step, we prove that the smallest l such that $WR_i^{(l)}(C_i) = WR_i^{(l+1)}(C_i)$ equals the smallest l such that $WR_i^{(l)}(C_i') = WR_i^{(l+1)}(C_i')$.

For the proof of the first step, we first prove the equation for $l = 0$, i.e.

$$\begin{aligned}
WR_i^{(0)}(C_i') &= \{\text{Lemma 4.8}\}\ \frac{C_i'}{1-U_{i-1}} \\
&= \frac{C_i + kS_{i-1}}{1-U_{i-1}} \\
&= WR_i^{(0)}(C_i) + \frac{kS_{i-1}}{1-U_{i-1}} \\
&= \{(6.4)\}\ WR_i^{(0)}(C_i) + kH_{i-1}.
\end{aligned}$$

Next, if we assume $WR_i^{(l)}(C_i') = WR_i^{(l)}(C_i) + kH_{i-1}$ (induction hypothesis), we get

$$\begin{aligned}
WR_i^{(l+1)}(C_i') &= \{(4.4)\}\ C_i' + \sum_{j<i} \left\lceil \frac{WR_i^{(l)}(C_i')}{T_j} \right\rceil C_j \\
&= \{\text{induction hypothesis}\} \\
&\quad\ C_i + kS_{i-1} + \sum_{j<i} \left\lceil \frac{WR_i^{(l)}(C_i) + kH_{i-1}}{T_j} \right\rceil C_j \\
&= \{T_j \mid H_{i-1} \text{ and } (6.4)\} \\
&\quad\ C_i + kH_{i-1}(1-U_{i-1}) + \sum_{j<i} \left\lceil \frac{WR_i^{(l)}(C_i)}{T_j} \right\rceil C_j + kH_{i-1}U_{i-1} \\
&= WR_i^{(l+1)}(C_i) + kH_{i-1},
\end{aligned}$$

which completes the proof. The proof of the second step follows immediately by means of substitution in the equation proven in the first step, i.e.

$$\begin{aligned} & WR_i^{(l)}(C_i') = WR_i^{(l+1)}(C_i') \\ \Leftrightarrow \quad & WR_i^{(l)}(C_i) + kH_{i-1} = WR_i^{(l+1)}(C_i) + kH_{i-1} \\ \Leftrightarrow \quad & WR_i^{(l)}(C_i) = WR_i^{(l+1)}(C_i). \end{aligned}$$

□

6.3 A quantitative analysis

We performed an experiment to compare the cost in terms of required execution time to determine worst-case response times using the standard initial value ι_i^{SW} and the combined initial value $\iota_i^{\mathrm{CW}} = \max(\iota_i^{\mathrm{SW}}, \iota_i^{\mathrm{AW}})$ quantitatively. In this section, we first define a cost function for an RTA test. Next, we define our experiment. We then present results, and finally draw conclusions.

6.3.1 Cost of an exact test

The definition of the *cost* ξ_i^τ of calculating the response time of task τ_i is based on the iterative procedure described in Section 4.2.3.

$$\xi_i^\tau(C_i, \iota_i) = \xi_{init} + \omega_i(C_i, \iota_i)(\xi_{fixed} + (i-1)\xi_{sum})$$

In this equation, ξ_{init} is the cost of calculating $WR_i^{(0)}$, and $(\xi_{fixed} + (i-1)\xi_{sum})$ is the cost of a single iteration including the evaluation of the termination condition. The term ξ_{sum} is the cost of evaluating $\left\lceil WR_i^{(l)}/T_j \right\rceil C_j$, and ξ_{fixed} represents a fixed part. The *cumulative cost* of tasks τ_1 till τ_i is denoted by $\xi_i(\iota)$, and defined as

$$\xi_i(\iota) = \sum_{j \le i} \xi_j^\tau(C_j, \iota_j).$$

The cost of evaluating the RTA test for task set $\mathcal{T}$ is denoted by $\xi(\iota)$. When the RTA test is performed for successive tasks in the set, $\xi(\iota)$ is defined as $\xi(\iota) = \xi_m(\iota)$, where m either identifies the first task τ_m for which the deadline D_m is exceeded or m equals n.

6.3.2 Experiment definition

Given (6.9) and Corollary 6.1, we may construct cases such that $\omega_i^{\mathrm{S}}(kS_{i-1})/\omega_i^{\mathrm{A}}(kS_{i-1}) \ge \kappa$ for arbitrary $\kappa \in \mathbb{N}^+$. Similarly, we may construct cases such that $\xi(\iota^{\mathrm{SW}})/\xi(\iota^{\mathrm{CW}}) \ge \kappa$ for arbitrary $\kappa \in \mathbb{N}^+$, which suggests a huge advantage for the usage of ι^{CW} compared to ι^{SW} for special cases.

As a starting hypothesis for our experiment, we assume the utilization factor U_n, the spread σ_n of the periods that is defined as $\sigma_n = {}^{10}\log T_n/T_1$, and the number of tasks n to have a major influence on the cost. We therefore conducted experiments with:

- U_n ranging from 0.2 till 1 with a step size of 0.2, using a uniform distribution for U_i^τ,
- σ_n ranging from 0.1 till 4 with a step size of 0.1, where $T_1 = 1$, using a uniform distribution for T_i with $1 < i < n$, and sorting T_i conform the RM algorithm (hence we assume $D_i = T_i$), and
- $n \in \{3, 4, 6, 10, 20\}$.

For every combination of U_n, σ_n, and n, we performed 10,000 experiments. Based on the initial results,similar experiments were subsequently conducted for U_n ranging from 0.8 till 1 with a step size of 0.05, and for U_n ranging from 0.96 till 1 with a step size of 0.01. Hence, we performed three main sets of experiments.

In the equation for the cost, the terms ξ_{init} and ξ_{fixed} are negligible (i.e. assumed to be zero), hence we approximate $\xi^\tau(C_i, \iota_i)$ by.

$$\xi_i^\tau(C_i, \iota_i) \approx \omega_i(C_i, \iota_i)(i-1)\xi_{sum}.$$

With the variables mentioned above, we should parameterize our cost function, i.e. $\xi(\iota, U_n(e), \sigma_n, T_i(e), n, e)$, where e denotes the experiment. To prevent excessive verbosity, we will use $\xi(\iota, e)$. The following values were determined by the experiment, amongst others:

- *average relative complexity*, which is given by:
$$arc = \frac{\sum_e \xi(\iota^{\mathrm{SW}}, e)}{\sum_e \xi(\iota^{\mathrm{CW}}, e)};$$
- *maximum relative complexity*, which is given by:
$$mrc = \max_e \frac{\xi(\iota^{\mathrm{SW}}, e)}{\xi(\iota^{\mathrm{CW}}, e)}.$$

6.3.3 Experimental results

Figure 6.5 shows the average relative complexity *arc* for $U_n = 0.9$ for all the different values of n. Notably, the average advantage of the combined initial value ι^{CW} reduces for increasing n. The same observation holds for other values of U_n. The maximum relative complexity *mrc* is shown in Figure 6.6 for $n = 3$. The value rises steeply for $\sigma_n \geq 1.3$ from $U_3 = 0.95$ to $U_3 = 1$, illustrating a major advantage of the usage of ι^{CW} for high utilizations. There is a peak for $U_3 = 1$ and $\sigma_3 = 3.6$ with a height of 1631, corresponding with $\xi(\iota^{\mathrm{SW}}) = 4893\xi_{sum}$, and hence $\xi(\iota^{\mathrm{CW}}) = 3\xi_{sum}$. Just like the average relative complexity *arc*, the maximum relative complexity *mrc* reduces for increasing n. As examples, for $n = 4$, there is a peak for $U_4 = 1$ with a

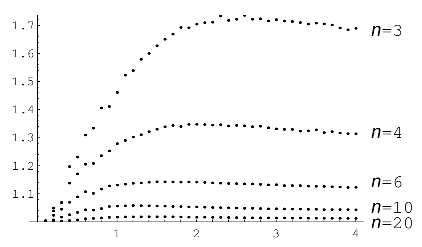


Figure 6.5. Average relative complexity *arc* for $U_n = 0.9$.

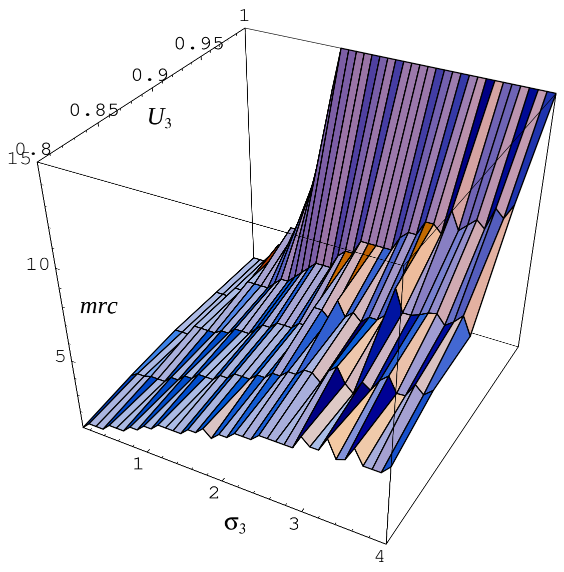


Figure 6.6. Maximum relative complexity *mrc* for $n = 3$.

height of 70, for $n = 6$, there is a peak for $U_6 = 1$ with a height of approximately 8.5, and for $n = 20$, *mrc* even stays below 1.2.

The *maximum complexity* mc^{CW} for ι^{CW} defined as $mc^{\mathrm{CW}} = \max_e \xi(\iota^{\mathrm{CW}}, e)$ is shown in Figure 6.7 for $n = 3$. This figure shows that mc^{CW} remains bounded, with a maximum of $48\xi_{sum}$ (for $U_3 = 0.95$ and $\sigma_3 = 3.9$). Notably, mc^{CW} drops steeply from $U_3 = 0.95$ to $U_3 = 1$.

6.3.4 Conclusions

Figure 6.5 shows that the average advantage of using ι^{CW} for a large number of task sets is only minor. Considering the peak of Figure 6.6, the advantage may however be substantial (a gain in cost of more than a factor 1000) for special cases. Moreover, the maximum value $\xi(\iota^{\mathrm{SW}}) = 4893\xi_{sum}$ for the peak of Figure 6.6 is

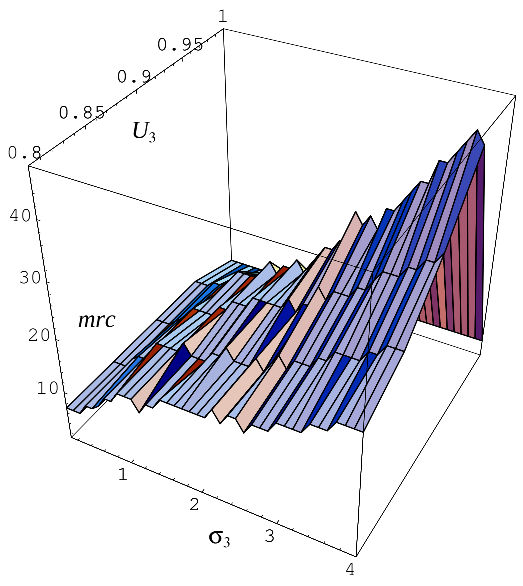


Figure 6.7. Maximum complexity *mc* in terms of the number of times ξ_{sum} for $n = 3$.

more than a factor 100 larger than the maximum $\xi(\iota^{\mathrm{CW}}) = 48\xi_{sum}$ in Figure 6.7. Hence, the main advantage of the usage of ι^{CW} is the reduction of the worst-case cost of performing an RTA test.

6.4 Discussion

6.4.1 Related work

Both initial values $WR_{i-1} + C_i$ and ι_i^{A} for determining worst-case response times have already been presented by Sjödin & Hansson [1998]. They also evaluate the benefits of these two values in the context of RTA tests for ATM networks, and conclude that compared to using $WR_{i-1} + C_i$ only, the effect of its combination with ι_i^{A} is negligible. Based on our experiment, we refine their result. Although the advantage of using the combination is only minor on average, our results show that the main advantage is found in the reduction of the worst-case cost of performing an RTA test.

Similar to our approach, the RTA test presented by Bini & Buttazzo [2002] also requires only a bounded number of iterations per task. They show that their test requires fewer iterations than a standard RTA test determining worst-case response times by means of an iterative procedure using $WR_{i-1} + C_i$ as initial value. However, unlike our approach, the test in that paper does not determine worst-case response times, and is therefore only applicable in situations where worst-case response times are not needed.

6.4.2 Calculating worst-case response times revisited

From the derivation of $WR_i^{(l)}(C_i') = WR_i^{(l)}(C_i) + kH_{i-1}$ in the proof of Theorem 6.3 we draw the following conclusion.

Corollary 6.2. *For arbitrary $C_i > 0$ and $k \in \mathbb{N}$ the following equation holds.*

$$WR_i(C_i + kS_{i-1}) = WR_i(C_i) + kH_{i-1} \tag{6.13}$$

□

There is an alternative approach based on Corollary 6.2 that also requires only a bounded number of iterations to determine $WR_i(C_i)$ for arbitrary C_i. Let M_i be the maximum number of iterations to calculate $WR_i(C_i)$ for $C_i \in (0, S_{i-1}]$. For arbitrary C_i, we define $C_i' = C_i - kS_{i-1}$, where $k = \lceil C_i/S_{i-1} \rceil - 1$, hence $C_i' \in (0, S_{i-1}]$. By calculating $WR_i(C_i)$ by means of $WR_i(C_i) = WR_i(C_i') + kH_{i-1}$, we therefore need at most M_i iterations. Note that this alternative approach comes at the additional cost of determining the hyperperiod H_{i-1} and the slack S_{i-1}.

6.4.3 Calculating best-case response times and occupied times

Similarly to $WR_i(C_i)$, $\iota_i^{\mathrm{A}}(C_i)$ increases with H_{i-1} when C_i increases with S_{i-1}, as shown in the derivation of $WR_i^{(0)}(C_i') = WR_i^{(0)}(C_i) + kH_{i-1}$ in the proof of Theorem 6.3.

Corollary 6.3. *For arbitrary $C_i \geq 0$ and $k \in \mathbb{N}$ the following equation holds.*

$$\iota_i^{\mathrm{A}}(C_i + kS_{i-1}) = \iota_i^{\mathrm{A}}(C_i) + kH_{i-1} \tag{6.14}$$

□

This property of $\iota_i^{\mathrm{A}}(C_i)$ is the cornerstone of the proof that $\omega_i^{\mathrm{A}}(C_i + kS_{i-1}) = \omega_i^{\mathrm{A}}(C_i)$ for $k \in \mathbb{N}$ in the proof of Theorem 6.3.

Because the same property holds for standard initial values to determine $BR_i(C_i)$, such as $WR_i(C_i)$ (see Corollary 6.2) and $WR_i(C_i) - WR_{i-1}$, usage of $\iota_i^{\mathrm{A}}(C_i)$ does not give a similar advantage as for worst-case response times. The main advantage of $\iota_i^{\mathrm{A}}(C_i)$ compared to these standard initial values is that is allows the calculation of $BR_i(C_i)$ without the need to calculate $WR_i(C_i)$ first.

Based on these observations, we state the following theorems for the number of iterations to determine the best-case response time, the best-case occupied time, and the worst-case occupied time. The proofs of these theorems are similar to the proof of Theorem 6.3.

Theorem 6.4. *The number of iterations $\beta_i(C_i)$ needed to calculate the best-case response time $BR_i(C_i)$ of task τ_i as a function of C_i for both a standard initial value, such as $WR_i(C_i)$ or $WR_i(C_i) - WR_{i-1}$, and the alternative initial value given by (5.15) is bounded by a constant and periodic with period S_{i-1}.* □

Theorem 6.5. *The number of iterations $\overline{\beta}_i(C_i)$ needed to calculate the best-case occupied time $BO_i(C_i)$ of task τ_i as a function of C_i for both a standard initial value, such as $WR_i(C_i)$, and the alternative initial value given by (5.15) is bounded by a constant and periodic with period S_{i-1}.* □

Theorem 6.6. *The number of iterations $\overline{\omega}_i(C_i)$ needed to calculate the worst-case occupied time $WO_i(C_i)$ of task τ_i as a function of C_i for either an initial value such as $WR_i(C_i)$ or the alternative initial value given by (5.15) is bounded by a constant and periodic with period S_{i-1}.* □

We mention that the number of iterations $\omega_i(C_i)$ needed to calculate the $WO_i(C_i)$ of τ_i as a function of C_i for a standard initial value, such as given by (4.12), also grows logarithmically.

Finally, we give a number of lemmas expressing similar properties for the worst-case occupied time, the best-case response time, and the best-case occupied time as the property given for the worst-case response time in Corollary 6.2. Note that the proofs of these lemmas are based on Corollary 6.2 and the properties of response times and occupied times for discrete scheduling.

Lemma 6.4. *For arbitrary $C_i \geq 0$, $k \in \mathbb{N}$, and T_i sufficiently large, the following equation holds.*

$$WO_i(C_i + kS_{i-1}) = WO_i(C_i) + kH_{i-1} \tag{6.15}$$

Proof. The proof uses (6.1) and (6.13).

$$\begin{aligned}
WO_i(C_i + kS_{i-1}) &= \{(6.1)\}\ WR_i(C_i + kS_{i-1} + 1) - 1 \\
&= \{(6.13)\}\ WR_i(C_i + 1) + kH_{i-1} - 1 \\
&= \{(6.1)\}\ WO_i(C_i) + kH_{i-1}
\end{aligned}$$

□

Lemma 6.5. *For arbitrary $C_i > 0$, $k \in \mathbb{N}$, and T_i sufficiently large, the following equation holds.*

$$BR_i(C_i + kS_{i-1}) = BR_i(C_i) + kH_{i-1} \tag{6.16}$$

Proof. The proof uses (6.8) and Lemma 6.4. For $l \in \mathbb{N}^+$ such that $T_i \geq lH_{i-1}$ and

$C_i + kH_{i-1} \le lS_{i-1}$, we can apply (6.8).

$$\begin{aligned} BR_i(C_i + kS_{i-1}) &= \{(6.8)\}\ lH_{i-1} - WO_i(lS_{i-1} - (C_i + kS_{i-1})) \\ &= lH_{i-1} - WO_i(lS_{i-1} - (C_i + kS_{i-1})) - kH_{i-1} + kH_{i-1} \\ &= \{\text{Lemma 6.4}\}\ lH_{i-1} - WO_i(lS_{i-1} - C_i) + kH_{i-1} \\ &= \{(6.8)\}\ BR_i(C_i) + kH_{i-1} \end{aligned}$$

□

Lemma 6.6. *For arbitrary $C_i \ge 0$, $k \in \mathbb{N}$, and T_i sufficiently large, the following equation holds.*

$$BO_i(C_i + kS_{i-1}) = BO_i(C_i) + kH_{i-1} \tag{6.17}$$

Proof. The proof is similar to the proof of Lemma 6.4 using (6.2) and (6.16). □

7

A Resource Manager for Conditionally Guaranteed Budgets

This chapter is concerned with CGBs (conditionally guaranteed budgets). It presents the concept of a CGB, and an extension of the existing resource manager with a mechanism to schedule CGBs. The additional admission test corresponding with this mechanism is the topic of Chapter 8.

We start this chapter with a real-time scheduling model for budgets in Section 7.1. This model is a simplified version of the task model presented in Section 3.1. Next, in Section 7.2, we present the concept of a CGB. As described before, a CGB facilitates an instantaneous budget configuration change. In order to come to grips with the corresponding requirements for a mechanism to schedule CGBs, we will first analyze budget configuration changes in Section 7.3. Next, we present the concept of in-the-place-of resource provision in Section 7.4. That concept provides the basis for a mechanism to schedule CGBs. The mechanism is the topic of Section 7.5, and is described in terms of extensions to the existing resource manager, the so-called budget scheduler. This chapter is concluded with a discussion on existing work, enhancements, and future work in Section 7.6.

7.1 A real-time scheduling model for budgets

In essence, a budget may simply be viewed as an artificial task, and all notions introduced for tasks in Chapter 3 can be reused for budgets. Our budget model is merely a simplified version of the task model presented in Section 3.1. The fact that budgets are allocated and provided to RCEs, which consist of one or more tasks, is irrelevant for the model. For explanatory purposes, we will first briefly introduce budgets and their consumption by means of tasks.

Hence, we assume a single processor and a set $\mathcal{B}$ of n independent periodic budgets, denoted by $\beta_1, \beta_2, \ldots \beta_n$. At any moment in time, the processor is used to 'execute' the highest priority budget that has not yet been depleted. Execution of a budget means consumption of the budget by an RCE to which that budget is provided. When a budget β_i is consumed by an RCE, the highest priority task that has work pending of that RCE is being executed by the processor, and that execution is accounted to β_i. When a budget β_i is being consumed, and a release occurs for a higher priority budget β_j, then the consumption of β_i is preempted, and will resume when the consumption of β_j has ended, as well as all other releases of budgets with a higher priority than β_i that have taken place in the meanwhile.

The introduction of budgets therefore gives rise to two levels of scheduling: scheduling of budgets and scheduling of tasks consuming budgets. In principle, the mechanisms used at each level can be chosen independently. In the existing implementation, budgets and tasks are both scheduled using FPPS, and the analysis presented in Chapter 8 also assumes FPPS of budgets.

Given the similarity between our task model and our budget model, we confine ourselves in this section to a description of budget characteristics in Section 7.1.1, and the assumptions we make on the environment in Section 7.1.2.

7.1.1 Budget characteristics

Each budget β_i is characterized by a (*release*) *period* $T_i \in \mathbb{R}^+$, an amount of time $B_i \in \mathbb{R}^+$ (which we also term a *budget*), a (*relative*) *deadline* $D_i \in \mathbb{R}^+$, and a *phasing* $\varphi_i \in \mathbb{R}$. The combination of phasings φ_i is termed the phasing φ of the budget set $\mathcal{B}$. Note that we do not differentiate between best-case budgets and worst-case budgets. The *activation* (*release* or *request*) *time* is the time at which a budget β_i becomes ready for execution, i.e. when the budget is replenished.

7.1.2 Basic assumptions

The following basic assumptions are made on the environment, for which the analytical results are obtained. Note that these assumptions are similar to those described in Section 3.1.

1. Budgets have unique priorities.

2. Consumption of a budget will be preempted instantaneously when a higher priority budget becomes ready for consumption.
3. Budgets are ready for consumption, i.e. they are replenished, at the start of each period and consumption is not suspended.
4. Budgets are independent, i.e. there is no budget synchronization.
5. The overhead of context switching and budget scheduling is ignored.
6. Activation $k+1$ of budget β_i does not start before the end of activation k.
7. No specific phasing of budgets at start-up is assumed.

Moreover, we assume that budgets are fixed and the deadlines are hard, i.e. when a budget is replenished upon its activation, it must be depleted before the corresponding deadline. Finally, we assume that deadlines of budgets are equal to their periods.

The schedulability of a set $\mathcal{B}$ of n periodic budgets is, similarly to tasks, given by (3.7).

7.2 The concept of a conditionally guaranteed budget

In this section, we present an outline of a solution for the user-focus problem in terms of CGBs, and describe two variants of CGBs. In order to come to grips with the problem, we will first analyze the reaction time in more detail. We subsequently describe a general model for CGBs, and then present the variants of CGBs.

7.2.1 Analysis of the reaction time

The reaction time (from time t_I to t_S in Figure 1.5 on page 12) is based on three succeeding actions: detecting the structural load increase (from t_I to t_D), determining the new mode (from t_D to t_A), and effectuating the new mode (from t_A to t_S). We now consider each of these three activities in more detail.

Detecting the structural load increase

There are two main options for detecting the structural load increase, by means of derivation from the data stream and through measurements during data processing.

As an example of the first option, the data stream may contain auxiliary data including estimates of induced load for processing the data stream or characteristics from which these estimates can be derived. Given such information, the structural load increase may even be determined *before* it appears, i.e. t_D may be smaller than t_I. Unfortunately, it is uncommon that media data in broadcast environments provide the necessary information to detect upcoming structural load changes timely. Coded media streams, such as MPEG-2 [Haskell et al., 1997], typically provide dedicated, so-called *user data* fields allowing the enhancements of those streams with proprietary information. For MPEG-2, information could be added to the headers of the entire stream, of the group-of-pictures and of the individual pictures. However, being based on proprietary information, such an approach only has its merits in closed environments. Alternatively, an application may determine the cause of the load change (like a scene or shot change, or a change from movie towards video) by means of dedicated processing entities, such as film detectors [De Haan, 1992]. Finally, an application may be triggered to change its mode of operation, and may have higher resource demands for the new mode.

When the structural load increase cannot be derived from the data stream, it may be detected through measurements during data processing. Local control algorithms, such as those described by Lafruit et al. [2000] and Wüst & Verhaegh [2002], perform this detection as a by-product of their normal activity. Detection by these algorithms can be fast, because they have semantic interpretation for the resource usage measures. Alternatively, when the measurements are done by another, semantically neutral entity, such as a wrapper or container component encapsulating the application [De Miguel, Ruiz & García, 2002], detection of the structural nature of the overload necessarily takes time. So, in that case we must seek the solution in eliminating the structural resource shortage.

Determining the new mode

As mentioned before, determining the new mode takes quite some time, and the load increase must therefore be anticipated. To this end, the QM needs to predetermine two modes of operation, a *normal mode* accommodating the normal load of the more important application (MIA) and an *anticipated mode* accommodating the anticipated load of MIA after the structural load increase. To accommodate the load increase, MIA receives a higher budget in the anticipated mode than in the normal mode. The higher budget of MIA can be accommodated by giving the less important application (LIA) a lower budget. Conversely, the lower budget of MIA in the normal mode allows for an additional budget for LIA in that mode compared to the anticipated mode. Thus, the budgets allocated to MIA and LIA, and the corresponding quality settings, anticipate the load increase for MIA, and both MIA and LIA are informed in advance about the existence of the two modes; see also Table7.1.

Effectuating the new mode

Upon a structural load increase, the higher budget for MIA must become available instantaneously. Conversely, the additional budget ΔB_{LIA} that LIA receives in the normal mode will be withdrawn instantaneously, and LIA may be confronted with a sudden reduction of its budget, rather than being informed timely. As a consequence, the additional budget that LIA receives in the normal mode can only

Table 7.1. An example of a normal mode and an anticipated mode for CGBs. Note that LIA receives a quality setting Q'_{LIA} for the normal mode and a quality setting Q_{LIA} for the anticipated mode.

	normal mode (low load for MIA)	anticipated mode (high load for MIA)
MIA	$Q_{\mathrm{MIA}}, B_{\mathrm{MIA}}$	$Q_{\mathrm{MIA}}, B_{\mathrm{MIA}} + \Delta B_{\mathrm{MIA}}$
LIA	$Q'_{\mathrm{LIA}}, B_{\mathrm{LIA}} + \Delta B_{\mathrm{LIA}}$	$Q_{\mathrm{LIA}}, B_{\mathrm{LIA}}$

be allocated with a *conditional guarantee*, i.e. it is guaranteed (just like normal budgets) under the condition that MIA does not need it. This additional budget is therefore termed a *conditionally guaranteed budget* (CGB).

When the QM allocates both an absolutely guaranteed budget B_{LIA} and a CGB ΔB_{LIA} to LIA, it informs LIA that it can run on a quality level Q_{LIA} matching the AGB, and that it *may* run on a quality level Q'_{LIA} matching the combination of budgets. In this way, the CGB provides an option for *controlled* quality improvement.

When confronted with a change in the availability of the CGB, LIA is assumed to adjust to the new situation by selecting the appropriate quality level. Smooth transitions may therefore not always feasible, and we have to revert to a best-effort approach for LIA. Hence, the structural load increase causing the user-focus problem still takes its toll, but we shifted the resulting problem to a place where it is less severe. Moreover, the application facing the problem may be selected for its ability to handle it.

7.2.2 A model for CGBs

Although CGBs are conceived as a means to solve the user-focus problem, the notion of user focus nor the notion of relative importance plays a role at the level of the budget scheduler. Rather than using terms like MIA and LIA, we therefore prefer terms that are independent of these notions. In the sequel, we use CGB provider ρ_p (rather than MIA) for the RCE that *provides* the CGB and CGB consumer ρ_c (rather than LIA) for the RCE that *consumes* the CGB.

Assume that ρ_p requires an AGB of B_p in normal mode, and an additional budget ΔB_p in anticipated mode. This additional budget ΔB_p is also termed a *budget margin*. Conversely, ρ_c requires an AGB of B_c in anticipated mode and is granted an additional budget ΔB_c with a conditional guarantee in normal mode.

Because the budget margin must become available to ρ_p instantaneously, we require that a mode change from normal mode to anticipated mode can be performed instantaneously. In a next section, we will present a mechanism that facilitates such an instantaneous mode change.

In the following subsections, we consider the two variants of CGBs. So-called *strong* CGBs can be applied when the structural load increase can be detected timely. Otherwise we have to eliminate the structural resource shortage, and we can apply *weak* CGBs.

7.2.3 Weak CGBs

Without appropriate means for detection, we must seek the solution in eliminating the structural resource shortage. Hence, rather than assigning an AGB B_p to ρ_p in the normal mode, we revert to the higher budget $B_p + \Delta B_p$ for ρ_p corresponding to its AGB in the anticipated mode. When the anticipated load increase occurs, it thus can be accommodated without delay. Due to the budget margin, ρ_p has a *budget surplus*, and therefore generates gain time before the load increase. When ρ_p consistently generates gain time corresponding with its budget margin, ρ_c consistently receives its CGB ΔB_c. Note that even when ρ_c consistently receives the CGB, peak loads of ρ_p may still cause occasional unavailability of the CGB. This type of CGB is therefore termed a *weak* CGB. To compensate ρ_c for the potential fluctuations in the provision of the CGB, we assume that *all* gain time of ρ_p is provided to ρ_c. Other alternatives are conceivable, but fall outside the scope of this document.

7.2.4 Strong CGBs

With appropriate means for detection, ρ_p can explicitly *release* and *claim* its budget margin, thereby instructing the budget scheduler to switch between both budget configurations. When ρ_p claims its budget margin in normal mode, ρ_c is informed that its CGB will be withdrawn instantaneously, followed by an instantaneous mode change. Between the release and subsequent claim of the budget margin, the CGB consumer can therefore count on its CGB. This type of CGB is therefore termed a *strong* CGB.

Note that a synchronous ρ_p will be able to restrain itself during every budget period. An asynchronous ρ_p cannot do so, precisely because its behavior is asynchronous with respect to its budget period. Without explicit help of the budget scheduler, an asynchronous ρ_p will keep consuming its budget margin until it blocks for lack of input.

Finally note that although the information about the withdrawal is provided timely, ρ_c will typically not be ready to receive the information when it is processing a unit of work. This may give rise to the quality degradation with overshoot shown in Figure 1.7.

7.3 Instantaneous budget configuration changes

For weak CGBs, ρ_p is always in possession of its budget margin ΔB_p, and ρ_c only receives its CGB ΔB_c when ρ_p consistently generates gain time corresponding with that budget margin. For strong CGBs, ρ_p receives ΔB_p in anticipated mode only.

In normal mode, ρ_p only receives a budget B_p. In order for ρ_p to receive its budget margin, a switch to anticipated mode is required, and to that end ρ_p needs to explicitly claim its budget margin. Upon its claim, that budget margin must become available instantaneously, and the CGB will therefore be withdrawn instantaneously. Hence, a mechanism to schedule CGBs must facilitate an instantaneous budget configuration change. However, that mechanism must not interfere with the guarantees for AGBs. In this section, we analyze budget configuration changes to come to grips with the requirements for a mechanism to schedule CGBs.

Figure 2.11 on page 35 shows a mode change involving two applications, including a budget configuration change (BCC). In this section, we first illustrate a BCC for FPPS in more detail. We subsequently show that an instantaneous BCC may cause deadline misses of AGBs under both FPPS and EDF.

7.3.1 Budget configuration change

To illustrate a BCC under FPPS, consider Figure 7.1, of which the characteristics are given in Table 7.2. Both budget configurations are schedulable under FPPS. We assume that the RCEs may run asynchronously with respect to their budget periods. The figure contains three time lines, denoted by (i) – (iii), each showing a particular variant of a BCC. For all three time lines, all four RCEs have a simultaneous release of their AGBs at time $t = 0$. Moreover, it is assumed that ρ_p has released its budget margin before time $t = 0$, and re-claims it at time $t = 14$, which gives rise to a BCC request at $t_{bcr} = 14$. In time line (i), the system waits for an idle interval and

Table 7.2. An example 'task set' to illustrate budget configuration changes under fixed-priority preemptive scheduling.

	period	AGB		CGB
RCE	T_i	B_i	ΔB_p	ΔB_c
ρ_1	8	2		
ρ_p	12	3	1	
ρ_c	48	1		4
ρ_4	60	19		

subsequently performs the BCC. As mentioned before, waiting for an idle interval ensures that the BCC will not affect the guarantees of budgets [Tindell & Alonso, 1996]. The first idle interval occurs at time $t = 111$; the CGB is withdrawn from ρ_c and the budget margin is provided to ρ_p, i.e. $t_{dec} = t_{inc} = 111$. The budget margin becomes available to ρ_p for the first time at $t = 125$, implying a delay of nine AGB periods of ρ_p, which is too long.

In time line (ii), the CGB is withdrawn immediately from ρ_c upon ρ_p's claim, i.e. $t_{dec} = t_{bcr} = 14$. The provision of the budget margin is once again delayed

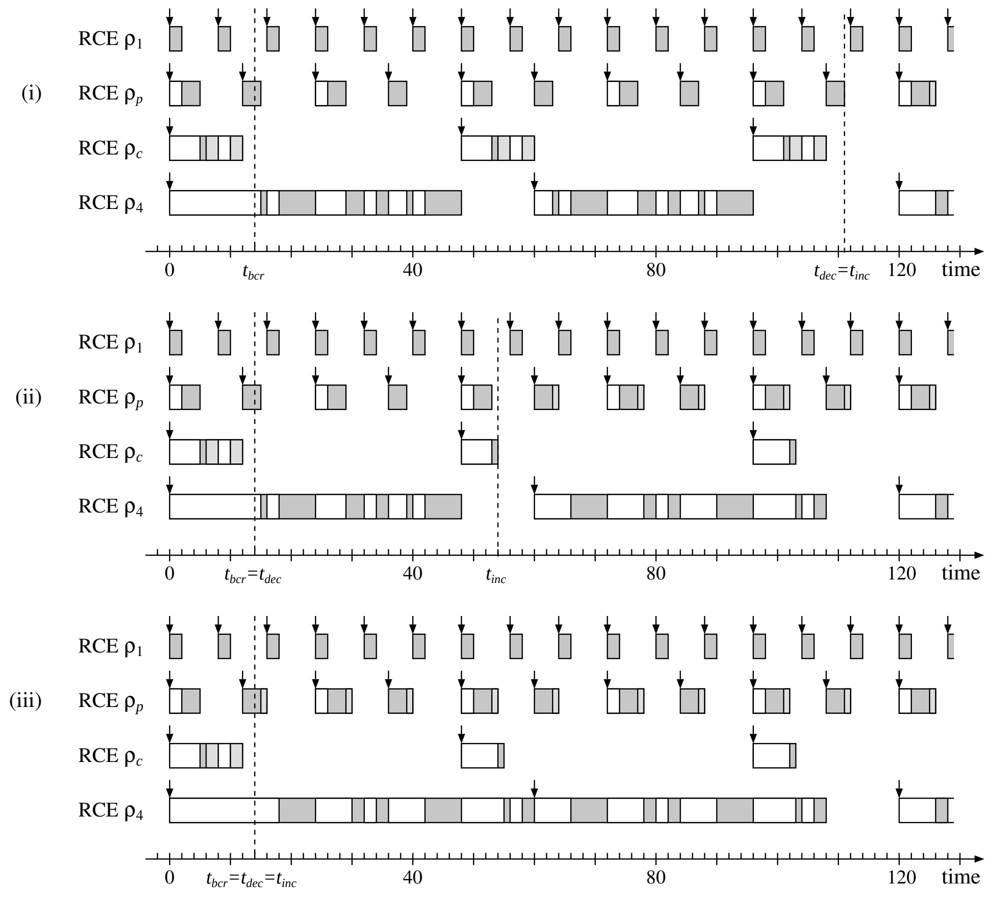


Figure 7.1. Three variants of BCCs for the 'task set' of Table 7.2. For all three, RCE ρ_p re-claims its budget margin $\Delta B_p = 1$ at time $t_{bcr} = 14$. Time line (i) shows a BCC with $t_{bcr} \ll t_{dec} = t_{inc}$. In time line (ii), the CGB is withdrawn immediately, and the AGB is increased later, i.e. $t_{bcr} = t_{dec} < t_{inc}$. Time line (iii) shows an instantaneous BCC, with $t_{bcr} = t_{dec} = t_{inc}$.

till the first idle interval, which occurs at time $t = 54$, i.e. $t_{inc} = 54$. The budget margin becomes available to ρ_p for the first time at $t = 63$, implying a delay of four AGB periods of ρ_p. Although this is a considerable improvement, it is still a substantial delay. Note that this variant gives rise to an additional *transitory* budget configuration in which the CGB has already been withdrawn from ρ_c, but the AGB of ρ_p has not been increased with the budget margin yet.

Time line (iii) shows that for this particular case the BCC can also be performed instantaneously without causing a deadline miss for any budget. For this particular case $t_{dec} = t_{inc} = t_{bcr} = 14$, and the budget margin can be used by ρ_p at time $t = 15$, i.e. in the same budget period in which ρ_p claimed it. In other cases, an instantaneous BCC may give rise to budget interferences, as illustrated in the following sections.

7.3.2 Budget interference under FPPS

Figure 7.2 shows an example of budget interference upon an instantaneous BCC under FPPS. The characteristics of Figure 7.2 can be found in Table 7.3. Both budget configurations are schedulable under FPPS. The budgets of all four RCEs

Table 7.3. An example 'task set' that gives budget interference under fixed-priority preemptive scheduling.

	period	AGB		CGB
RCE	T_i	B_i	ΔB_p	ΔB_c
ρ_1	8	2		
ρ_p	12	3	2	
ρ_c	24	1		4
ρ_4	30	6		

are released simultaneously at time $t = 0$, and ρ_p has released its budget margin before that time. When ρ_p subsequently re-claims its budget margin at time $t_{bcr} = 14$, and the BCC is performed instantaneously, the AGB of ρ_4 misses its deadline at time $t = 30$.

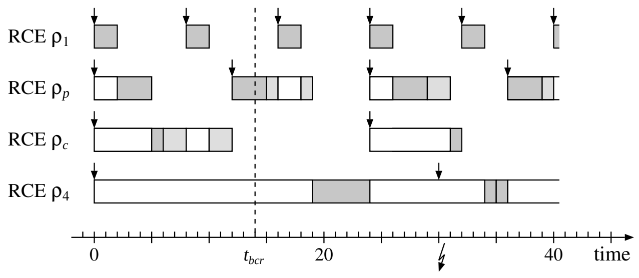


Figure 7.2. Budget interference under FPPS for the 'task set' of Table 7.3. RCE ρ_p re-claims its budget margin $\Delta B_p = 2$ at time $t_{bcr} = 14$. The budget margin is provided instantaneously, causing a deadline miss of the AGB of ρ_4 at $t = 30$.

For this particular example, the CGB of ρ_c corresponds to the budget margin of *two* budget periods of ρ_p, i.e. $\Delta B_c = 4 = 2\Delta B_p$. Hence, the problem is caused by the fact that the CGB is provided to ρ_c *before* the corresponding budget margins would have been provided to ρ_p. Withdrawing the CGB of ρ_c at time $t_{bcr} = 14$ would yield an idle interval starting at time $t = 23$. Hence, when the provision of the budget margin to ρ_p would be delayed by one period, ρ_4 would not miss its deadline.

7.3.3 Budget interference under EDF

Figure 7.3 shows a similar example of budget interference upon an instantaneous BCC under EDF. The characteristics of Figure 7.3 can be found in Table 7.4. Both budget configurations are schedulable under EDF. This particular example shows

Table 7.4. An example 'task set' that gives budget interference under earliest deadline first.

	period	AGB		CGB
RCE	T_i	B_i	ΔB_p	ΔB_c
ρ_c	16	4		8
ρ_p	20	5	10	

that it is impossible to provide the entire budget margin in the same period as it is claimed. Moreover, when the remaining time of the budget period of ρ_p is provided to ρ_p, a deadline miss of the AGB of ρ_c occurs at time $t = 80$.

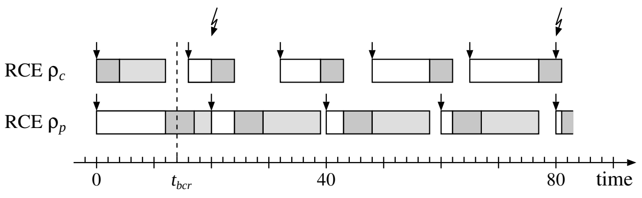


Figure 7.3. Budget interference under EDF for the 'task set' of Table 7.4. RCE ρ_p re-claims its budget margin $\Delta B_p = 10$ at time $t_{bcr} = 14$. The remaining time till the end of ρ_p's budget period is too short to provide the entire budget margin. When the remaining time is provided to ρ_p, a deadline miss of the AGB of ρ_c occurs at $t = 80$.

7.3.4 Analysis

Figures 7.2 and 7.3 illustrate deadline misses when the budget margin is provided instantaneously to ρ_p upon its re-claim. In both cases the CGB has been provided to RCE ρ_c *before* the budget margins on which the CGB is based would have been provided to ρ_p. Hence, when the budget margin is re-claimed by ρ_p, it may no longer be (completely) available during its current budget period. Note that Figure 7.1 shows that there also exist situations where the CGB is provided before the corresponding budget margins and this problem does not occur.

7.4 The concept of in-the-place-of resource provision

The problem of interfering budgets can be solved by making sure that the CGB is not provided to RCE ρ_c before the budget margin would have been provided to ρ_p. This can be accomplished by applying the so-called concept of *in-the-place-of resource provision* for CGB provisioning, i.e. the CGB is provided to ρ_c *at* the period and phasing of the budget of ρ_p, and *when* the budget margin would have been provided to ρ_p. This concept of *in-the-place-of CGB provision* generalizes the so-called *in-the-place-of* design that was originally conceived for weak CGBs [Otero Pérez, 2001], and applies to both FPPS and EDF.

In the following subsections we illustrate in-the-place-of CGB provision for strong CGBs for both FPPS and EDF. The examples show the following three characteristics of the concept. Firstly, the CGB may be withdrawn *in the midst of* its provision during a budget period. Secondly, its application may result in *scheduling imperfections*, even for EDF. Hence, immediateness of the availability of the budget margin ΔB_p to ρ_p upon its re-claim may reduce the amount of CGB ΔB_c. Finally, the CGB may become available with *absolute jitter* [Klein, Ralya, Pollak, Obenza & González Harbour, 1993].

7.4.1 Example for FPPS

To illustrate in-the-place-of CGB provision for FPPS, consider Figure 7.4, using the characteristics of Table 7.3. The figure contains two time lines, denoted by (i) and (ii). For both time lines, all four RCEs have a simultaneous release of their AGBs at time $t = 0$, and it is assumed that ρ_p has released its budget margin before that time.

Time line (i) shows that the CGB is provided to ρ_c during the time intervals that the budget margin would have been provided to ρ_p. Time line (i) also shows that the budget margin ΔB_p becomes available with (absolute) *jitter* due to preemptions by ρ_1. In time line (ii), ρ_p re-claims its budget margin at time $t_{bcr} = 14$. The CGB is withdrawn from ρ_c and the budget margin is provided to ρ_p during the current budget period. Note that the CGB is withdrawn in the midst of its provision.

7.4.2 Example for EDF

To illustrate in-the-place-of CGB provision for EDF, consider Figure 7.5, using the characteristics of Table 7.4. The figure contains five time lines, denoted by (i) – (v). For all five time lines, both RCEs have a simultaneous release of their AGBs at time $t = 0$.

For time line (i), it is assumed that ρ_p has claimed its budget margin before time $t = 0$. Based on the in-the-place-of CGB provision, we derive from that time line that ρ_c could consume 7 units of time in the first three periods, 9 units of time in the fourth period, and 10 units of time in the fifth period. It is fairly easy to

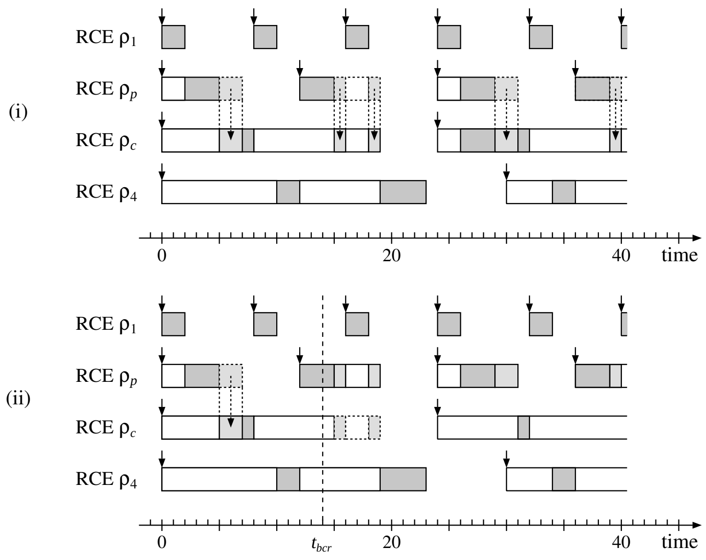


Figure 7.4. Time lines with examples for in-the-place-of CGB provision under FPPS for the 'task set' of Table 7.3. Time line (i) shows that the CGB becomes available to ρ_c when the budget margin ΔB_p would have come available to ρ_p. Time line (ii) shows that when ρ_p re-claims its budget margin at time $t_{bcr} = 14$, the CGB is withdrawn in the midst of its provision.

determine the amount $\Delta B_c^{\mathrm{avg}}$ of CGB that can be conditionally given *on average* based on the budget margin, because the utilization of the *CGB*$_c$ and the budget margin are the same, i.e.

$$\Delta B_c^{\mathrm{avg}} = T_c \frac{\Delta B_p}{T_p}. \tag{7.1}$$

Note that $\Delta B_c^{\mathrm{avg}}$ is also an *upper bound* for the amount that can be conditionally guaranteed on a periodic basis. For this example, $\Delta B_c^{\mathrm{avg}} = 8$. Hence, although 8 units of time can be provided as CGB to ρ_c on average, at most 7 units of time can be guaranteed on a strictly periodic basis. Applying the in-the-place-of CGB provision for EDF therefore results in *spare capacity* due to *scheduling imperfections*. Time line (i) also shows that the budget margin becomes available with *jitter*.

Assume ρ_c receives a CGB with a strictly periodic budget $\Delta B_c = 7$. Time lines (ii) – (iv) illustrate different alternatives to deal with the gain time corresponding to the remainder of the budget margin. For all three time lines, ρ_p has released its budget margin before $t = 0$. In time line (ii), the entire budget margin ΔB_c is provided as CGB to ρ_c. Hence, ρ_c is (implicitly) allowed to consume all gain time corresponding to the remainder of the budget margin. In time line (iii), ρ_c

is constrained to a periodic CGB of 7 units of time. The remainder of the budget margin becomes available when both RCEs have consumed their budgets, i.e. *as late as possible*. It is left to a spare-capacity manager to provide the slack to either of the RCEs. ρ_c is also constrained to a strictly periodic CGB $\Delta B_c = 7$ in time line (iv). In this case, the spare capacity becomes available when ρ_c has depleted its CGB and there is still budget margin left, i.e. during the time intervals that it is *produced*. Classification of this spare capacity as either *slack time* or *gain time* may depend on the particular perspective taken. It may be classified as slack time from a 'scheduling imperfection' point of view, and as gain time from a 'remainder-of-the-budget-margin' point of view. Although one may prefer the former view for strong CGBs and the latter for weak CGBs, we rather treat both on equal footing, and therefore use the term gain time in the remainder. Note that the provision of gain time at the time it is produced may be viewed as an application of the concept of in-the-place-of resource provision for gain-time provisioning. We will denote this specialization of this general concept by *in-the-place-of gain-time provision*.

For time line (v), it is assumed that ρ_p has released its budget margin before time $t = 0$. ρ_p re-claims its budget margin at time $t_{bcr} = 26$. The CGB is withdrawn from ρ_c and the budget margin is provided to ρ_p during its current budget period. Hence, the CGB is withdrawn in the midst of its provision, similarly to the example for FPPS that was shown in time line (ii) in Figure 7.4.

7.5 A mechanism for scheduling conditionally guaranteed budgets

The implementation of CGBs strongly depends on the implementation of the budget scheduler (BS). We therefore start with a concise description of the basic implementation of the BS. More advanced implementation issues are addressed in Section 7.6.1. We subsequently describe a conceivable extension of the BS with in-the-place-of gain-time provision. Finally, we briefly describe a possible extension of the BS with in-the-place-of CGB provision for weak and strong CGBs.

We characterize specific aspects of the implementation of budgets by means of invariants, in order to allow an easy comparison.

7.5.1 Budget scheduler

The basic implementation of the BS has been described in Section 2.5. Below, we therefore only characterize the implementation by means of invariants.

Let B_i^{var} denote the amount of AGB still available to RCE ρ_i in the current budget period, hence $0 \leq B_i^{\text{var}} \leq B_i$. Concerning priority manipulations, the implementation should maintain the following invariants. The phrase 'ρ_i did not stop' denotes that ρ_i neither blocked nor released the CPU. Similarly, the phrase 'ρ_i

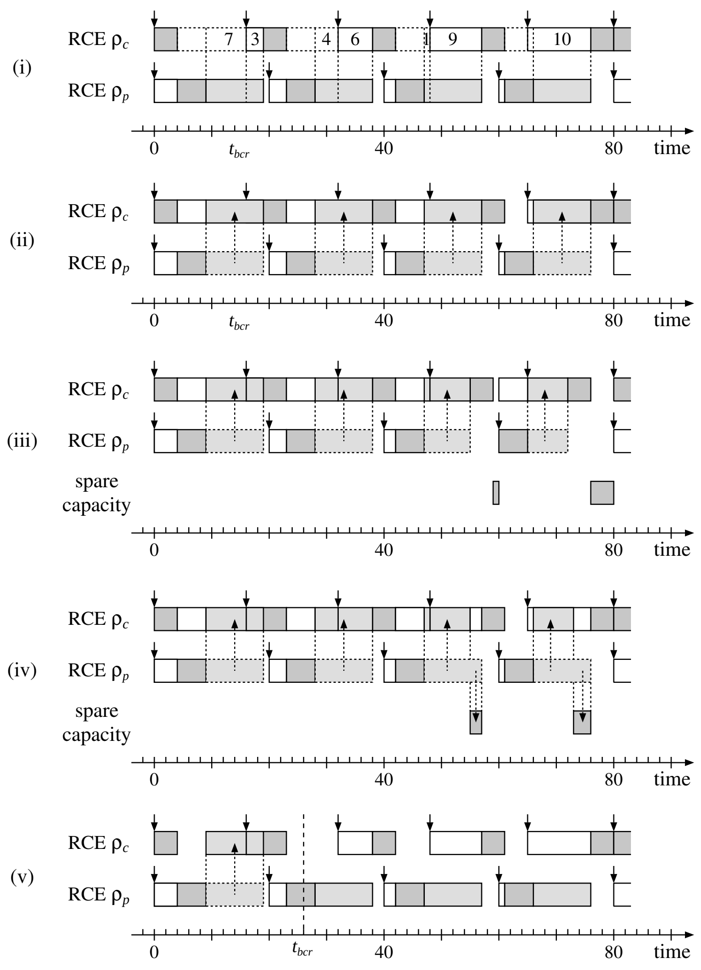


Figure 7.5. Time lines with examples for the in-the-place-of CGB provision under EDF for the 'task set' of Table 7.4. Time line (i) shows that in-the-place-of CGB provision results in *scheduling imperfections*. Time lines (ii) till (iv) show three variants for handling the resulting spare-time. In time line (v), ρ_p re-claims its budget margin at time $t_{bcr} = 26$. The instantaneous BCC causes a withdrawal of the CGB *in the midst* of its provision.

stopped' denotes that ρ_i either blocked or released the CPU.

$$\rho_i \text{ at AGB priority in HP} \;\Leftrightarrow\; B_i^{\text{var}} > 0 \wedge \rho_i \text{ did not stop}$$

$$\rho_i \text{ in LP} \;\Leftrightarrow\; B_i^{\text{var}} = 0 \vee \rho_i \text{ stopped}$$

7.5.2 Extension with in-the-place-of gain-time provision

In the existing implementation, the gain time of RCEs only becomes available for out-of-budget execution in LP, i.e. *as late as possible*. We therefore call this mechanism *latest* gain-time provision. As mentioned before, the existing implementation does not facilitate policies for explicit allocation of spare capacity, nor does it account out-of-budget executions.

Experience revealed the following problem caused by withdrawal of the budget upon blocking as implemented in the BS. A streaming RCE, such as an MPEG-2 decoder, may privately use a dedicated hardware device, such as a variable length decoder. After the RCE requests the device to perform an action, it has to wait for (i.e. it blocks till) the completion of that device before it can resume its execution. Indiscriminately withdrawing a budget from a streaming RCE upon blocking therefore gives rise to problems. For this thesis, we therefore argue in favor of a less stringent implementation of gain-time production upon blocking. Upon blocking, only the time an RCE could have executed but is blocked is accounted to its budget. Moreover, the priority of the RCE remains unaltered upon blocking. When the RCE un-blocks and there is still AGB left, the RCE can resume its execution. Other kinds of blocking are not supported by the BS, and also fall outside the scope of this thesis. The interested reader is referred to [De Niz, Saewong, Rajkumar & Abeni, 2001; De Niz, Abeni, Saewong & Rajkumar, 2001].

As a basis for our in-the-place-of gain-time mechanism, we view a budget as a server, and associate a budget with a range of sub-priority bands in HP. A single RCE *owns* the budget and performs in-budget execution at its sub-priority band in HP. The sub-priority band of the RCE that owns the budget is the highest band within the range of sub-priority bands of that budget. Apart from the RCE that *owns* the budget, *every* RCE with a sub-priority band in the range of the budget is allowed to run on that budget. An RCE therefore effectively *shares* its budget with all RCEs having a sub-priority band in the range of its budget. This is illustrated in Figure 7.6. Hence, when the owning RCE blocks or releases the processor,

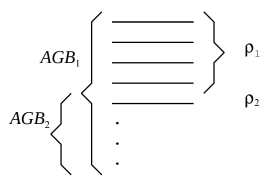


Figure 7.6. A budget as a server. RCE ρ_1 *owns* AGB_1, and *shares* it will all RCEs with a lower sub-priority band in the priority range of AGB_1.

the RCE with the highest priority lower than the owning RCE is allowed to execute on the owning RCE's budget till either the latter releases the processor, the owning RCE un-blocks or the budget is exhausted. In summary, at any time the execution of the RCE with the highest priority is accounted to the highest priority (non-depleted) budget, where the system maintains an invariant guaranteeing that the sub-priority band of the executing RCE falls inside the priority range of the accounted budget. This basic mechanism only facilitates default (i.e. implicit) gain-time provision, and is therefore called *implicit* in-the-place-of gain-time provision. Note that the replacement of the latest gain-time provision mechanism by the implicit in-the-place-of gain-time provision mechanism influences budget accounting and influences priority manipulations of RCEs upon blocking The invariants for in-the-place-of gain-time provision are

$$\begin{aligned} \rho_i \text{ at AGB priority in HP} \quad &\Leftrightarrow \quad B_i^{\text{var}} > 0 \wedge \rho_i \text{ did not release the CPU} \\ \rho_i \text{ in LP} \quad &\Leftrightarrow \quad B_i^{\text{var}} = 0 \vee \rho_i \text{ released the CPU.} \end{aligned}$$

As a refinement, we may introduce a sub-priority band P' for gain-time provision for every sub-priority band P in HP, where P' is positioned immediately below P. Whereas P corresponds with an RCE owning a budget, P' is exclusively meant for gain-time provision. A spare capacity manager can subsequently explicitly allocate the gain time of the owner of a budget associated with P to an RCE by raising the priority of that RCE to P'. This mechanism is called *explicit* in-the-place-of gain-time provision. We consider priority manipulations in more detail when we discuss the extensions for CGBs in the next sections. Additional refinements and more sophisticated mechanisms are conceivable, but fall outside the scope of this thesis.

7.5.3 Extension with in-the-place-of CGB provision for weak CGBs

As mentioned before in Section 7.2.3, we assume that *all* gain time of the CGB provider ρ_p will be provided to ρ_c for weak CGBs, i.e. gain time from both B_p as well as ΔB_p. We therefore can apply the same approach as described for explicit in-the-place-of gain-time provision to implement in-the-place-of CGB provision for weak CGBs, i.e. reserve an additional sub-priority band in HP immediately below the sub-priority band of the AGB of the CGB provider ρ_p, and executions of ρ_c at CGB priority in HP are accounted to the AGB of ρ_p.

We now consider the required priority manipulations for ρ_c for this mechanism. For ease of presentation, we assume implicit in-the-place-of gain-time provision and ignore the release of the CPU by ρ_c. To this end, we distinguish two main cases, based on the priorities of the AGBs of ρ_p and ρ_c, and four situations, corresponding with the start and depletion of the AGBs of ρ_p and ρ_c.

For the first case, the priority of the AGB of ρ_c is lower than the priority of the AGB of ρ_p, i.e. in-budget executions of ρ_c are performed at a lower sub-priority band in HP than of ρ_p.

At the start of a new period of the AGB of ρ_p, B_p^{var} is set to $B_p + \Delta B_p$, and the priority of ρ_c is raised to its CGB provision sub-priority level in HP. When the AGB of ρ_p is exhausted (i.e. $B_p^{\text{var}} = 0$), the priority of ρ_c is lowered to its AGB level when ρ_c has still AGB left (i.e. $B_c^{\text{var}} > 0$) and otherwise to LP. At the start of a new period of the AGB of ρ_c, B_c^{var} is set to B_c, and the priority of ρ_c is raised to its AGB provision sub-priority level in HP if and only if it had a priority in LP. When the AGB of ρ_c is exhausted (i.e. $B_c^{\text{var}} = 0$), the priority of ρ_c is lowered to LP. Note that because the AGB of ρ_c exhausted, it must have been the highest non-depleted budget, and the priority of ρ_c could therefore not be higher than corresponding to its AGB. We characterize this case for the priority of ρ_c by

$$\begin{aligned}
\rho_c \text{ at CGB priority in HP} &\Leftrightarrow B_p^{\text{var}} > 0 \\
\rho_c \text{ at AGB priority in HP} &\Leftrightarrow B_p^{\text{var}} = 0 \wedge B_c^{\text{var}} > 0 \\
\rho_c \text{ in LP} &\Leftrightarrow B_p^{\text{var}} = 0 \wedge B_c^{\text{var}} = 0.
\end{aligned}$$

For the second case, the priority of the AGB of ρ_c is higher than the priority of the AGB of ρ_p. At the start of a new period of the AGB of ρ_p, the priority of ρ_c is raised to its CGB provision sub-priority level in HP if and only if it had a priority in LP. When the AGB of ρ_p is exhausted, the priority of ρ_c is returned to LP. Note that because the AGB of ρ_p exhausted, it must have been the highest non-depleted budget, and the priority of ρ_c could therefore not be higher than that of the AGB of ρ_p. At the start of a new period of the AGB of ρ_c, the priority of ρ_c is raised to its AGB provision sub-priority level in HP. When the AGB of ρ_c is exhausted, the priority of ρ_c is lowered to the AGB of ρ_p when the latter has still budget left and otherwise to LP. The characterizing invariants are

$$\begin{aligned}
\rho_c \text{ at AGB priority in HP} &\Leftrightarrow B_c^{\text{var}} > 0 \\
\rho_c \text{ at CGB priority in HP} &\Leftrightarrow B_c^{\text{var}} = 0 \wedge B_p^{\text{var}} > 0 \\
\rho_c \text{ in LP} &\Leftrightarrow B_c^{\text{var}} = 0 \wedge B_p^{\text{var}} = 0.
\end{aligned}$$

7.5.4 Extension with in-the-place-of CGB provision for strong CGBs

Unlike weak CGBs, the gain time of ρ_p is not the basis for strong CGBs. For strong CGBs, a CGB is treated as a special kind of budget, giving rise to a dedicated sub-priority band in HP with an accompanying sub-priority band for explicit gain-time provision. That band is positioned immediately below the sub-priority band of the AGB of ρ_p and its accompanying sub-priority band for explicit gain-time provision. Unlike weak CGBs, executions of ρ_c at CGB in HP are accounted to the CGB of ρ_c rather than the AGB of ρ_p. Hence, although weak CGBs and strong CGBs are variants of the same concept, and although they share the application of the concept of in-the-place-of CGB provision for their implementations, their actual implementations differ. As mentioned above, the gain time of ρ_p is not the basis for strong CGBs. Hence, the concepts strong CGBs and gain-time provision are orthogonal, and although their implementations are both based on the general concept of in-the-place-of resource provision, they are treated independently at the level of implementation.

For the description of the implementation, we assume that the CGB has to be provided at a periodic basis. Hence, ρ_c receives a CGB of at most ΔB_c in normal mode, allowing the remainder of ΔB_p to become available for gain-time provision. Let ΔB_p^{var} denote the amount of the budget margin ΔB_p still available in the current budget period of ρ_p, and ΔB_c^{var} denote the amount of ΔB_c still available in the current budget period of ρ_c. ρ_c is allowed in-budget executions at CGB priority in HP if and only if *both* ΔB_p^{var} and ΔB_c^{var} are larger than zero, and those executions are accounted to both ΔB_p^{var} and ΔB_c^{var}. Gain-time executions accounted to the CGB are only feasible when ΔB_p^{var} is larger than zero, are accounted to both ΔB_p^{var} and ΔB_c^{var}, and only decrease ΔB_c^{var} as long as it is larger than zero. Other aspects of the implementation are similar to that of weak CGBs, and we therefore only briefly describe the required priority manipulations and other additional changes for the four situations below for the case where the priority of the absolutely guaranteed budget of ρ_c is lower than the priority of the absolutely guaranteed budget of ρ_p. Moreover, we also assume implicit in-the-place-of gain-time provision in our description for ease of presentation.

We will now look at the two modes, anticipated and normal, and the changes between both modes. In the anticipated mode, the CGB is not provided, priority manipulations for CGB provision are therefore not needed, and the dedicated sub-priority bands for the CGB remain unused. The invariants for the anticipated mode are therefore the same as for the extension with in-the-place-of gain-time provision.

In the normal mode, the CGB is provided. As mentioned before, we only consider the case where the priority of the absolutely guaranteed budget of ρ_c is lower than the priority of the absolutely guaranteed budget of ρ_p. At the start of a new period of the AGB of ρ_p, the ΔB_p^{var} of the CGB is set to ΔB_p, and the priority of ρ_c is raised to its CGB provision sub-priority level in HP if and only if ΔB_c^{var} is larger than zero. Exhaustion of the AGB of ρ_p has no implications for the priority of ρ_c. When either ΔB_p^{var} or ΔB_c^{var} of the CGB of ρ_c becomes zero, the priority of ρ_c is lowered to its AGB level when there is still budget left and otherwise to LP. At the start of a new period of the AGB of ρ_c, the ΔB_c^{var} of the CGB is set to ΔB_c. The priority of ρ_c is raised to its CGB provision sub-priority level in HP if ΔB_p^{var} is larger than zero and otherwise to its AGB level in HP. When the AGB of ρ_c is exhausted, the priority of ρ_c is lowered to LP. Note that because the AGB of ρ_c exhausted, it must have been the highest non-depleted budget, and the priority of

ρ_c could therefore not be higher than its AGB. The characterizing invariants are

$$\begin{aligned}
\rho_c \text{ at CGB priority in HP} &\Leftrightarrow (\Delta B_p^{\text{var}} > 0 \wedge \Delta B_c^{\text{var}} > 0)\\
\rho_c \text{ at AGB priority in HP} &\Leftrightarrow (\Delta B_p^{\text{var}} = 0 \vee \Delta B_c^{\text{var}} = 0) \wedge B_c^{\text{var}} > 0\\
\rho_c \text{ in LP} &\Leftrightarrow (\Delta B_p^{\text{var}} = 0 \vee \Delta B_c^{\text{var}} = 0) \wedge B_c^{\text{var}} = 0.
\end{aligned}$$

When ρ_p claims its budget margin, the remaining ΔB_p^{var} is added to its AGB, both ΔB_p^{var} and ΔB_c^{var} are set to zero, and the priority of ρ_c is lowered accordingly. Note that the entire budget margin ΔB_p is only guaranteed for claims performed by ρ_p during in-budget executions. This is because any execution is always accounted to the AGB of ρ_p before it is accounted to the budget margin of ρ_p. As a result, when ρ_p executes on spare capacity, its claim will not necessarily result in the provision of the entire budget margin in its current budget period.

For simplicity, we assume that when ρ_p releases its budget margin, the BCC from anticipated mode to normal mode will be effectuated at the start of a new budget period of ρ_p.

7.6 Discussion

In this section, we will revisit the existing implementations, expand on CGBs and their applicability, and describe subjects for further research.

7.6.1 Existing implementations revisited

Section 2.5 only described the basic implementation of the BS, and ignored prior designs and implementations of weak CGBs. In this section, we will consider the existing implementations in more detail and present a solution to an open problem.

Prior designs and implementations of weak CGBs

Bril & Steffens [2001] describe a first straightforward extension of the BS with weak CGBs, where the CGB is provided as a special kind of budget in a so-called middle-priority band MP between HP and LP. There are three drawbacks of that solution, however. Firstly, and most importantly, the CGB is consumed at a non-RM priority. Hence, no CGB can be guaranteed at ρ_c's period T_c when the lowest priority AGB in HP has a response time longer than T_c. Although the mechanism to modify periods presented by Sha, Lehoczky & Rajkumar [1986] solves this problem, this would add an undesired complexity to the system. Secondly, when the budget margin ΔB_p is used by ρ_p, the CGB does not become available, and ρ_c therefore claims spare capacity. This results in an undesired interference of CGB provision and spare-capacity allocation. Finally, this solution relies heavily on the use of RM priority assignment as a scheduling mechanism.

To circumvent the above drawbacks, the notion of *in-place* budget sharing was introduced in the context of FPPS by Otero Pérez, Bril & Steffens [2001]. In-the-place-of provision of the CGB to ρ_c is described in that paper as an extension of the existing implementation of the BS. Such a provision of CGB means that the priority of ρ_c is changed to the AGB priority of ρ_p in HP, and that the remaining AGB budget is transferred to the CGB of ρ_c. For a priority of the AGB of ρ_c lower than the priority of the AGB of ρ_p, the implementation can be briefly described as follows. When ρ_p releases the processor or blocks, its priority is lowered to LP. If the remaining amount of AGB of ρ_p is larger than zero, it is made available to the CGB of ρ_c, the AGB is depleted, and the priority of ρ_c is raised to ρ_p's AGB priority in HP. When the CGB is exhausted, the priority of ρ_c is lowered to its AGB in HP when there is still budget left or to LP otherwise. The following invariants incorporate this behavior.

$$\begin{aligned}
\rho_c \text{ at CGB priority in HP} \quad &\Leftrightarrow \quad (B_p + \Delta B_p)^{\text{var}} > 0 \wedge \rho_p \text{ stopped} \\
\rho_c \text{ at AGB priority in HP} \quad &\Leftrightarrow \quad ((B_p + \Delta B_p)^{\text{var}} = 0 \vee \rho_p \text{ did not stop} \\
&\qquad \wedge B_c^{\text{var}} > 0 \wedge \rho_c \text{ did not stop} \\
\rho_c \text{ in LP} \quad &\Leftrightarrow \quad ((B_p + \Delta B_p)^{\text{var}} = 0 \vee \rho_p \text{ did not stop}) \\
&\qquad \wedge (B_c^{\text{var}} = 0 \vee \rho_c \text{ stopped})
\end{aligned}$$

The main drawback of this approach for weak CGBs is that it cannot properly handle situations with blocking.

Gain-time production upon blocking, triggered CGBs, and time-offsets

Now consider a system with latest gain-time provision, and accounting for blocking as described above. An inadvertent implementation of strong CGBs by means of a dedicated sub-priority in HP as described in Section 7.5 would give rise to the following problem in such a system.

Assume the CGB provider ρ_p becomes blocked. Moreover, assume the CGB consumer ρ_c is at CGB in HP. In such a situation, ρ_c is allowed to execute despite the fact that the AGB of ρ_p has not been exhausted yet. When ρ_p un-blocks, is allowed to resume execution, and re-claims its budget margin, parts of the budget margin have already been consumed by ρ_c. As a consequence, an instantaneous BCC can no longer provide the entire budget margin without jeopardizing the guarantees of other AGBs.

This problem can be circumvented by so-called *triggered* strong CGBs, i.e. the CGB is triggered (enabled) when the AGB of ρ_p is exhausted rather than when the AGB of ρ_p is started. An alternative approach to solve this problem is to split the AGB of the CGB provider ρ_p in two separate budgets: a budget β_p for the normal budget B_p and an additional budget β'_p for the budget margin ΔB_p, where β'_p has a relative time-offset of at least $WR_p(B_p)$ with respect to β_p. By basing the strong

CGB on β'_p only, it can never be consumed too early. Because we confine ourselves in this thesis to a basic real-time scheduling model without time-offsets, we do not consider this latter approach in more detail.

Note that this problem does not occur for in-the-place-of gain-time provision, because the AGB of ρ_p is always depleted by either executions of ρ_p or gain-time executions before ρ_c is allowed to consume its CGB.

7.6.2 CGB variants

This section presents various extensions to the basic concept and alternatives to its mechanism.

Extensions of the concept

In this chapter, we presented a basic mechanism for CGBs, covering only a single anticipated load increase, and pairs of a single CGB provider and a single CGB consumer. Extensions of this basic mechanism for CGBs are conceivable, such as covering multiple levels of anticipated load increases, and pairs of multiple CGB providers and multiple CGB consumers.

Instantaneous and deferred budget configuration changes

In this chapter, we assumed the need for an instantaneous BCC. There also exist situations where a BCC can be deferred, allowing the CGB consumer to smoothly reduce its quality level. We will give two examples of options for a deferred BCC in this section.

As a first example, a load increases may be announced, e.g. by means of proprietary information embedded in the data stream in closed environments. As a second example, consider a load increase due to a scene or shot change. As mentioned before, the quality level may be dropped temporarily upon such a change, e.g. to alleviate an overload, because the human brain needs time to adjust. Moreover, when this time to adjust exceeds the lead-time to detect the load increase, the BCC can be deferred with the time surplus.

7.6.3 Application in other contexts

In this chapter, CGBs have been presented as a solution to the so-called user-focus problem. However, CGBs can be applied in other contexts as well, and the fact that CGBs facilitate an instantaneous BCC allows other types of applications to become a CGB provider as well. In this section, we show how an emergency application and an input reader for video data can serve as CGB provider. We mention that neither of these two applications is either scalable or adaptive, and that neither of them has a dedicated local controller to detect a structural load increase.

Emergency applications

Consider an emergency application, consisting of two main modes, *emergency detection* and *emergency handling*. By nature, an emergency is expected to happen infrequently, and the application is therefore expected to mainly run in emergency detection mode, making it a candidate for CGB provision. When the resource needs in both modes are known, and the mode change from detection to handling is required to be instantaneous, the load increase can be anticipated, and strong CGBs can be applied. Note that for this particular example the application does not need to be able to adjust its resource use to its budget, nor does it need a dedicated local controller to detect the load increase. Moreover, emergency applications may not need a normal budget in emergency detection mode, i.e. may have a normal budget equal to zero. Depending on the type of emergency, either an instantaneous BCC may actually be required for this type of application or a deferred BCC may be possible as well.

Input reader for video data

Consider an input reader for video data. Such an application cannot afford to miss input data, is typically neither adaptive nor scalable, and therefore requires a worst-case budget. When the input reader consistently uses fewer resources than its worst-case budget, its budget surplus can be exploited to improve the cost-effectiveness using controlled quality improvements of other RCEs by means of weak CGBs. Otherwise we have to revert to a mechanism for spare-capacity provision to exploit the budget surplus of the input reader.

7.6.4 Future work

Currently, there only exists an initial implementation for weak CGBs, and the existing implementation of the BS only supports latest gain-time provision.

Although we performed a number of initial experiments, the actual validation of CGBs by media processing applications is still a subject of further research, to be performed in close co-operation with our colleagues from the high-quality video domain. Moreover, the focus of this chapter has been on CGBs as a mechanism. The exploitation of this mechanism by policies residing in the controllers within the control hierarchy requires future work. Policies for cost-effective spare-capacity allocation are the topic of a pending patent publication.

Finally, in-the-place-of resource provision is considered to be a suitable approach to accommodate instantaneous BCCs under FPPS and an applicable approach under EDF. Other approaches are subject of future work, in particular for EDF.

8

An Admission Test for Conditionally Guaranteed Budgets

This chapter is concerned with the worst-case available amount ΔB_c that can be conditionally guaranteed for strong CGBs on a strictly periodic basis under arbitrary phasing for in-the-place-of resource provision of CGBs and gain time for FPPS. We present techniques to determine both optimistic and pessimistic bounds as well as exact values for that amount ΔB_c. These techniques use the extensions to RMA presented in Chapters 4 and 5.

We start this chapter with the notion of advancement in Section 8.1. This notion, which may be viewed as the inverse of the response time, eases the formulation of the techniques in subsequent sections. Next, in Section 8.2, we refine our budget model for CGBs, and present an example that will be used in the remainder of the chapter. Bounds for ΔB_c are the topic of Section 8.3. A technique to determine the exact amount of ΔB_c is described in Section 8.4, and an efficient algorithm to calculate ΔB_c is presented in Section 8.5. We conclude the chapter with a discussion in Section 8.6.

8.1 Advancement

In this section, we present the notion of advancement. We start with an informal introduction of the notion, and illustrate it by means of the example from Chapter 3. Next, we define the best-case advancement and worst-case advancement, and express them in terms of the best-case and worst-case response time, respectively. Both notions are also illustrated by the same example. Finally, we describe how to determine the best-case advancement and worst-case advancement.

8.1.1 Introduction

The *advancement* $A_i(t)$ of a task τ_i in an interval of time of length t starting at a release of τ_i is the amount of time that τ_i has executed in that interval. We will use an execution of task τ_3 as shown in Figure 3.3 for illustration purposes. Figure 8.1 shows both an execution of τ_3 and the advancement $A_3(t)$ for that execution. Note that the graph of $A_3(t)$ consists of diagonal line segments with a gradient of one and horizontal line segments. The diagonal line segment correspond to executions of τ_3, whereas the horizontal line segments correspond to preemptions of higher priority tasks or the completion of τ_3. Note that the advancement of a task τ_i may be viewed as the inverse of the response time of that task.

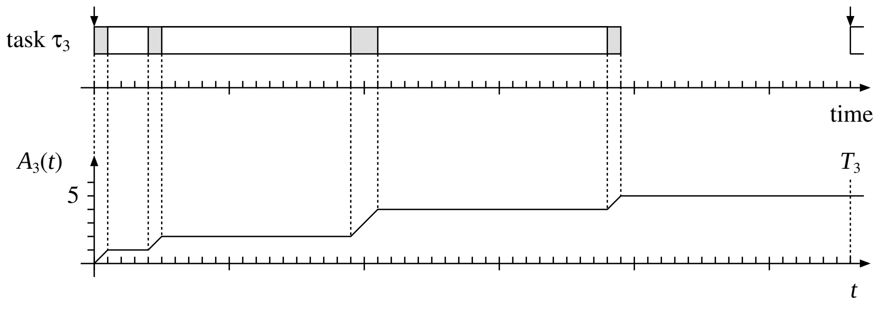


Figure 8.1. An execution of τ_3 and its advancement $A_3(t)$.

8.1.2 Best-case and worst-case advancement

The *best-case advancement* $BA_i(t)$ of a task τ_i in an interval of time of length t starting at a release of τ_i is the maximal amount of time that τ_i can execute in that interval. $BA_i(t)$ is given by

$$BA_i(t) = \begin{cases} \max\{C \mid BR_i(C) \leq t\} & \text{if } t > 0 \\ 0 & \text{if } t \leq 0. \end{cases} \tag{8.1}$$

Similarly, the *worst-case advancement* $WA_i(t)$ of a task τ_i in an interval of time of length t starting at a release of τ_i is the minimal amount of time that τ_i can execute in that interval. $WA_i(t)$ is given by

$$WA_i(t) = \begin{cases} \min\{C \mid WR_i(C) \leq t\} & \text{if } t > WO_i(0) \\ 0 & \text{if } t \leq WO_i(0). \end{cases} \tag{8.2}$$

From these definitions, we immediately see that $BA_i(t) \geq WA_i(t)$ for all t. Moreover, because $BR_i(C)$ and $WR_i(C)$ are monotonic increasing functions of C, $BA_i(t)$ and $WA_i(t)$ are monotonic non-decreasing functions of t.

The best-case advancement $BA_3(t)$ and worst-case advancement $WA_3(t)$ of task τ_3 of our example are shown in Figure 8.2. Executions for the best-case response times and worst-case response times of τ_3 have been shown in Figures 5.3 and 4.3, respectively. Note that the best-case execution has to be 'reversed' for the best-case advancement. The horizontal line-segments of $WA_3(t)$ in Figure 8.2 correspond to

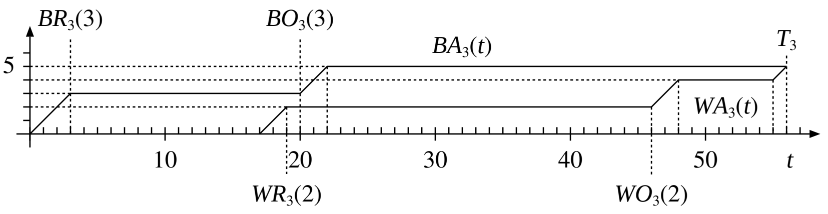


Figure 8.2. Best-case advancement $BA_3(t)$ and worst-case advancement $WA_3(t)$.

computation times C for which $WR_3(C)$ differs from $WO_3(C)$. As an example, $WR_3(2) = 19$ and $WO_3(2) = 46$. Note that the start of such a line segment corresponds to a worst-case response time and the end to a worst-case occupied time. We therefore term the start point of a horizontal line segment of a WA-graph a WO-bending point en the end point a WR-bending point.

Similar observations hold for the line segments of $BA_3(t)$.

8.1.3 Determining advancement graphs

The graphs for $BA_i(t)$ and $WA_i(t)$ are characterized by their bending points, and we can therefore determine an advancement graph by calculating its bending points. We will illustrate how bending points in the graphs for $BA_i(t)$ and $WA_i(t)$ can be calculated for the example given above.

We mention that the technique to determine bending points for $WA_i(t)$ is similar to the technique to determine spare capacity as described by Klein et al. [1993]. We start at $t = 0$, and observe that $WA_3(0) = 0$ by definition. The graph for $WA_3(t)$ has a horizontal line-segment for $0 \leq t \leq WO_3(0)$. The first bending point of $WA_3(t)$ is therefore found for $t = WO_3(0)$, and this bending point is termed a WO-bending point. The next bending point of $WA_3(t)$ is a WR-bending point, which is found at the end of a diagonal line-segment that starts at $WO_3(0)$, i.e. which is found at $t = WR_3(2)$. This point can be determined using a so-called first scheduling point

[Klein, Ralya, Pollak, Obenza & González Harbour, 1993]. Based on a critical instant, the *first scheduling point* $P_i^{\mathrm{FS}}(t)$ is the moment of the first release of any task with a higher priority than τ_i at or after time t, and is given by

$$P_i^{\mathrm{FS}}(t) = \min_{j<i} \lceil t/T_j \rceil T_j. \tag{8.3}$$

Given a *WO*-bending point at time $WO_i(C)$, the next *WR*-bending point $(WR_i(y), y)$ is found for $t = WR_i(y) = P_i^{\mathrm{FS}}(WO_i(C))$, where y is given by

$$y \;=\; C + P_i^{\mathrm{FS}}(WO_i(C)) - WO_i(C).$$

For our example, the *WR*-bending point after $WO_3(0)$ is characterized by

$$\begin{aligned} P_i^{\mathrm{FS}}(WO_3(0)) &= \min\{\lceil WO_3(0)/T_1\rceil T_1, \lceil WO_3(0)/T_2\rceil T_2\} \\ &= \min\{\lceil 17/10\rceil 10, \lceil 17/19\rceil 19\} = \min\{20,19\} = 19 \\ y &= 0 + P_i^{\mathrm{FS}}(WO_3(0)) - WO_3(0) = 0 + 19 - 17 = 2. \end{aligned}$$

Hence, we get a diagonal line segment of length 2. A horizontal line segment of $WA_i(t)$ that starts at $t = WR_i(y)$ ends at $t = WO_i(y)$. The latter value is determined by means of a standard technique. For our example, we derive $WO_3(2) = 46$, giving a horizontal line segment of length 27.

The technique to calculate the bending points of a *BA* graph is similar to that of a *WA* graph. The major differences being that it is based on a notion of next scheduling point rather than first scheduling point, and that the bending points are determined from above rather than from below. Based on an optimal instant, the *next scheduling point* $P_i^{\mathrm{NS}}(t)$ is the moment of the first release of any task with a higher priority than τ_i at or before time t, and is given by

$$P_i^{\mathrm{NS}}(t) = \max_{j<i} \lfloor t/T_j \rfloor T_j. \tag{8.4}$$

Given a *BR*-bending point at time $BR_i(C)$, the previous *BO*-bending point $(BO_i(y), y)$ is found for $t = BO_i(y) = P_i^{\mathrm{NS}}(BR_i(C))$, where y is given by

$$y \;=\; C + P_i^{\mathrm{NS}}(BR_i(C)) - BR_i(C).$$

For our example, the *BO*-bending point before $BR_3(5)$ is characterized by

$$\begin{aligned} P_i^{\mathrm{NS}}(BR_3(5)) &= \max\{\lfloor BR_3(5)/T_1\rfloor T_1, \lfloor BR_3(5)/T_2\rfloor T_2\} \\ &= \max\{\lfloor 22/10\rfloor 10, \lfloor 22/19\rfloor 19\} = \max\{20,19\} = 20 \\ y &= 5 + P_i^{\mathrm{NS}}(BR_3(5)) - BR_3(5) = 5 + 20 - 22 = 3. \end{aligned}$$

Hence, we get a diagonal line segment of length 2. The horizontal line segment that ends at $t = BO_i(y)$ starts at $t = BR_i(y)$. The latter value is determined by means of a standard technique. For our example, we derive $BR_3(3) = 3$, giving a horizontal line segment of length 17.

8.2 A refined budget model

As described in Section 7.1, our budget model is merely a simplified version of the task model presented in Section 3.1. Because a budget may simply be viewed as an artificial task, all the techniques presented in Chapters 4 and 5 can be reused for the real-time analysis of budgets. Remember that budgets are fixed and deadlines of budgets are hard, i.e. when a budget is replenished upon its activation, it must be depleted before the corresponding deadline. This latter requirement conforms to the in-the-place-of gain-time provision described in Section 7.4.

We will use a single example throughout the remainder of this chapter. Our example has a single CGB provider ρ_p and a single CGB consumer ρ_c, and is based on strong CGBs. In anticipated mode, an absolutely guaranteed budget AGB_p is provided to ρ_p, and that AGB_p is characterized by a budget consisting of a normal budget B_p plus the budget margin ΔB_p, a period T_p, and a phasing φ_p. Hence, when AGB_p is allocated to ρ_p, an amount of time $B_p + \Delta B_p$ will become available to ρ_p for execution on a strictly periodic basis. Similarly, an absolutely guaranteed budget AGB_c is provided to the CGB consumer ρ_c, and the AGB_c is characterized by a normal budget B_c, a period T_c, and a phasing φ_c.

In normal mode, i.e. the CGB provider ρ_p did not claim the budget margin ΔB_p, an absolutely guaranteed budget AGB'_p is provided to ρ_p. AGB'_p is characterized by the normal budget B_p, a period T_p, and a phasing φ_p. In this mode, a conditionally guaranteed budget CGB_c is allocated to ρ_c in addition to the absolutely guaranteed budget AGB_c. CGB_c is characterized by an additional budget ΔB_c (which we also term conditionally guaranteed budget), a period T_c, and a phasing φ_c. The additional budget ΔB_c is based on the budget margin ΔB_p from the absolutely guaranteed budget AGB_p. Note that CGB_c has the same period T_c and phasing φ_c as AGB_c. In total, ρ_c therefore receives a budget $B_c + \Delta B_c$ at a period T_c and a phasing φ_c in normal mode.

The relative phasing φ_R of AGB_c with respect to AGB_p is given by

$$\varphi_\mathrm{R} = (\varphi_c - \varphi_p) \bmod T_p. \qquad (8.5)$$

Note that $0 \le \varphi_\mathrm{R} < T_p$. For harmonic periods, i.e. $T_c = mT_p$ for $m \in \mathbb{N}^+$, the relative phasing of every activation of AGB_c is equal to φ_R. Stated in other words, the relative phasing is constant. For arbitrary periods, the relative phasing $\varphi_{\mathrm{R},q}$ of a particular activation q of AGB_c differs for each activation. So, the relative phasing of each activation of AGB_c is equal to φ_R if and only if T_c is an integral multiple of T_p.

The budget characteristics of AGB_p, AGB'_p and the higher priority AGBs of our example are given in Table 8.1. Given this information, we want to determine the highest lower bound of the amount that can be conditionally guaranteed to a CGB

Table 8.1. Budget characteristics of AGBs allocated to RCEs. RCE ρ_p is CGB provider.

	period	budget	
RCE	T_i	B_i	ΔB_p
ρ_1	6	1	
ρ_2	14	2	
ρ_p	29	9	7

consumer ρ_c. Note that ρ_c is not shown in Table 8.1.

8.3 Bounds for worst-case available budget margin

As mentioned before, the CGB allocated to a CGB consumer ρ_c is based on the budget margin allocated to the CGB provider ρ_p. As illustrated by means of Figure 7.5 on page 120, the CGB that can be guaranteed on a periodic basis under arbitrary phasings is generally smaller than the amount that can be conditionally given *on average* (7.1) due to scheduling imperfections.

The worst-case CGB ΔB_c that can be conditionally guaranteed on a strictly periodic basis and under arbitrary phasing is the highest lower bound of the amount of budget margin ΔB_p that becomes available in an interval of length T_c with an arbitrary phasing. We therefore consider ΔB_c as a function of T_c, and not of φ_R. Note that ΔB_c is at least zero, and is a strictly non-decreasing function of T_c. In this section, we determine optimistic and pessimistic bounds for $\Delta B_c(T_c)$. Exact values for $\Delta B_c(T_c)$ are the topic of the next section.

In order to arrive at bounds for ΔB_c, we sub-divide its domain into a number of sub-domains. To this end, we first determine the maximum value for T_c for which $\Delta B_c(T_c) = 0$. This maximum value is termed the *exact upper bound* $T_c^{\mathrm{EU}}(0)$. We next consider values of T_c larger than $T_c^{\mathrm{EU}}(0)$. For these values of T_c, we characterize situations with worst-case availability of budget margin. We subsequently show that the shape of the graph of $\Delta B_c(T_c)$ is periodic with period T_p, and present initial bounds for $\Delta B_c(T_c)$.

Next we tighten the bounds for $\Delta B_c(T_c)$. We therefore first present the minimum value for T_c for which $\Delta B_c(T_c) = \Delta B_p$, which we term *exact lower bound* $T_c^{\mathrm{EL}}(\Delta B_p)$. Because there is no simple algorithm to determine $T_c^{\mathrm{EL}}(\Delta B_p)$, we subsequently present a pessimistic alternative for $T_c^{\mathrm{EL}}(\Delta B_p)$. Finally, we present improved bounds for $\Delta B_c(T_c)$.

8.3.1 Exact upper bound

For $x \in \mathbb{R}$ and $x \geq 0$, the function $T_c^{\mathrm{EU}}(x)$ is defined as

$$T_c^{\mathrm{EU}}(x) = \sup\{T_c \mid \Delta B_c(T_c) = x\}. \tag{8.6}$$

In the remainder of this document, the term T_c^{EU} will be used as a shorthand for $T_c^{\mathrm{EU}}(0)$. The next lemma provides a means to determine T_c^{EU}.

Lemma 8.1. *The exact upper bound T_c^{EU} is given by*

$$T_c^{\mathrm{EU}} = T_p + WO_p(B_p) - BR_p(B_p + \Delta B_p). \tag{8.7}$$

Proof. The proof consists of two parts. In the first part, we characterize a situation in which we can find T_c^{EU}. We subsequently derive (8.7) by construction.

T_c^{EU} is the longest T_c-interval during which no budget margin becomes available. We therefore subdivide a period T_p, i.e. a T_p-interval, in three sub-intervals with respect to the availability of ΔB_p: a sub-interval *before*, *during*, and *after* ΔB_p becomes available. The length of the first sub-interval is at most the worst-case occupied time of B_p, i.e. $WO_p(B_p)$. Because $WO_p(B_p)$ is at least the worst-case start-time, i.e. $WO_p(B_p) \geq WO_p(0)$, the largest interval of joining preemptions of higher priority budgets embedded in the second sub-interval (during ΔB_p) is never larger than the first sub-interval. The length of the last sub-interval is at most the period T_p minus the best-case response time of $B_p + \Delta B_p$, i.e. $T_p - BR_p(B_p + \Delta B_p)$. The longest T_c-interval with no budget margin occurs when this last sub-interval in one period T_p joins this first sub-interval in the next period T_p. Hence, a situation with the longest T_c-interval in which no budget margin becomes available is characterized by a best-case advancement of the AGB_p in one period T_p immediately followed by a worst-case advancement of the AGB_p in the next period T_p.

We now derive (8.7) by construction. Consider two successive releases of AGB_p, and assume a best-case advancement $BR_p(B_p + \Delta B_p)$ for the first release immediately followed by a worst-case advancement of $WO_p(B_p)$ for the second release; see for example Figure 8.3. The largest T_c for which no budget margin becomes available in a T_c-interval is found when that T_c-interval starts at $BR_p(B_p + \Delta B_p)$ and ends at $T_p + WO_p(B_p)$. Hence, we find that $T_c^{\mathrm{EU}} = T_p + WO_p(B_p) - BR_p(B_p + \Delta B_p)$. □

For our example, we find $T_c^{\mathrm{EU}} = 29 + 16 - 21 = 24$. Note that T_c^{EU} can be larger than T_p, i.e. $T_c^{\mathrm{EU}} > T_p$ for $WO_p(B_p) > BR_p(B_p + \Delta B_p)$. This situation occurs for our example when we assume a B_p of 3 and a ΔB_p of 2, i.e. $BR_p(5) = 5$, $WO_p(3) = 7$, and therefore $T_c^{\mathrm{EU}} = 29 + 7 - 5 = 31$, which is larger than T_p.

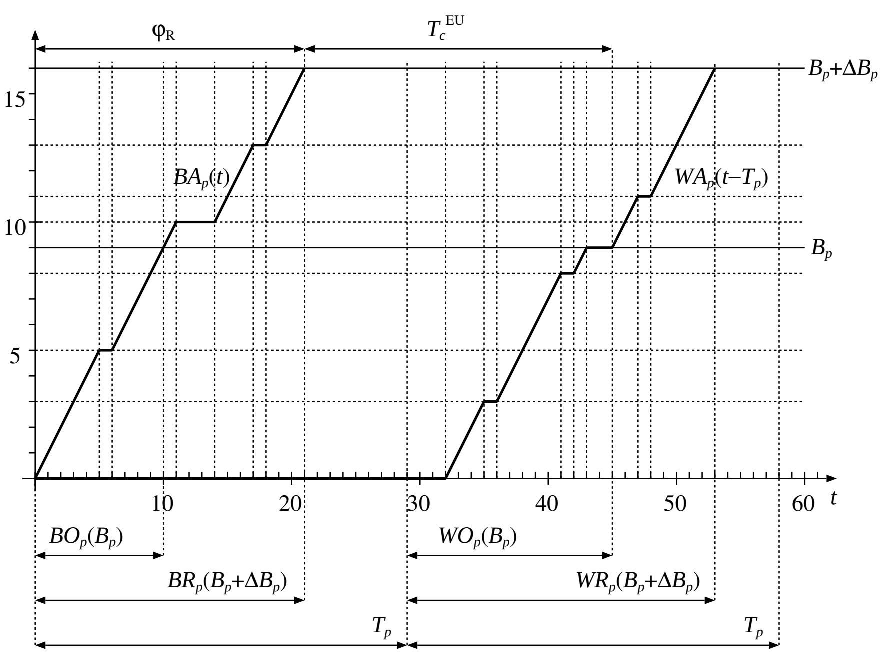


Figure 8.3. A best-case advancement of a budget in one period immediately followed by a worst-case advancement of the budget in the next period.

8.3.2 Situations of worst-case provision

For arbitrary periods T_p and T_c, we are (only) interested in the worst-case available conditionally guaranteed budget ΔB_c under arbitrary phasing. In that case, ΔB_c is the minimum worst-case available budget margin in a T_c-interval over all possible phasings of that interval. If we denote the amount of budget margin that becomes available for the activation with a specific relative phasing φ_R by $\Delta B_c(\varphi_R, T_c)$, then we may derive $\Delta B_c(T_c)$ by means of

$$\Delta B_c(T_c) = \inf_{\varphi_R} \Delta B_c(\varphi_R, T_c). \tag{8.8}$$

A T_c-interval corresponding with the release of AGB_c with relative phasing φ_R may have an overlap with multiple T_p-intervals of budget AGB_p. For periods T_p and T_c and activations of AGB_p and AGB_c with a relative phasing φ_R, the number l of T_p-intervals overlapping with a T_c-interval is given by

$$l = \lceil (\varphi_R + T_c)/T_p \rceil \tag{8.9}$$

and the relative phasing φ'_R of the next activation of AGB_c is given by

$$\varphi'_R = \varphi_R + T_c - \lfloor (\varphi_R + T_c)/T_p \rfloor T_p = (\varphi_R + T_c) \bmod T_p. \tag{8.10}$$

Given an AGB_p, we will describe a situation in which a minimal amount of budget margin becomes available in a T_c-interval with a relative phasing φ_R with respect to the activation of AGB_p. Such a situation describes the minimal provision (or availability) of the budget margin, and is therefore termed a *situation of worst-case provision*. When the number of overlapping intervals l is at least 2, the characterization of the worst-case provision is independent of the specific relative phasing φ_R, and the topic of the next lemma. We subsequently show that given arbitrary periods T_p and T_c, $l = 1$, and a particular phasing φ_R we can always construct a situation with $l = 2$ and phasing ϕ in which at most the same amount of budget margin becomes available in T_c. Stated in other words, the case $l = 1$ is dominated by $l = 2$ for the analysis of CGBs.

Lemma 8.2. *For $l \geq 2$, a worst-case provision for a particular relative phasing φ_R occurs when AGB_p experiences a best-case advancement BA_p in the first of those l T_p-intervals, and when AGB_p experiences a worst-case advancement WA_p in the last of those intervals.*

Proof. First of all, note that a worst-case provision is characterized by the first and last overlapping T_p-interval. Because the budget margin that becomes available in the remaining $l-2$ intervals is always covered by T_c, both the number of overlapping T_p-intervals and the exact time-intervals in which the budget margin becomes availability are irrelevant for a worst-case provision. Now consider the first of those overlapping T_p-intervals. The amount of budget margin ΔB_p that becomes available is minimal when ΔB_p becomes available as early as possible. This is the case for a best-case advancement BA_p of AGB_p. Finally, consider the l-th overlapping T_p-interval. The amount of budget margin ΔB_p that becomes available is minimal when ΔB_p becomes available as late as possible. This is the case for a worst-case advancement WA_p of AGB_p. □

For $l = 1$, we get $\varphi_R + T_c \leq T_p$, hence either $T_c = T_p \wedge \varphi_R = 0$ or $T_c < T_p \wedge \varphi_R \geq 0$. Obviously, $\Delta B_c = \Delta B_p$ for $T_c = T_p \wedge \varphi_R = 0$. Because we already covered the case $T_c \leq T_c^{\mathrm{EU}}$ in the previous section, we restrict ourselves to $T_c^{\mathrm{EU}} < T_c < T_p \wedge \varphi_R \geq 0$ here. Before presenting our lemma about domination, we first consider the special case where $BR_p(B_p + \Delta B_p) = T_p$.

Lemma 8.3. *For $T_c > T_c^{\mathrm{EU}}$, $l = 1$, and $BR_p(B_p + \Delta B_p) = T_p$, the worst-case available budget margin is found for a relative phasing $\varphi_R = 0$, and equal to $WA_p(T_c) - B_p = T_c - B_p$.*

Proof. Because we assume deadlines of budgets to be equal to their periods, and that they are met, $WR_p(B_p + \Delta B_p) \leq T_p$. By definition, $BR_p(B_p + \Delta B_p) \leq$

$WR_p(B_p + \Delta B_p)$. From $BR_p(B_p + \Delta B_p) = T_p$, we therefore derive that ρ_p's budget is the highest priority budget. The highest priority budget is consumed without preemptions, hence $B_p + \Delta B_p = T_p$. For $T_c^{\mathrm{EU}} < T_c \le T_p$ and $l = 1$, the worst-case available budget margin for a relative phasing φ_{R} is therefore given by $\varphi_{\mathrm{R}} + T_c - B_p$. This value is minimal for $\varphi_{\mathrm{R}} = 0$. □

For this trivial case, we can easily construct a situation with $l = 2$ and where the worst-case available budget margin remains the same. This can be done by applying the same construction as applied in the proof of Lemma 4.7 for worst-case execution times, i.e. move the T_c-interval from a situation where $\varphi_{\mathrm{R}} = 0$ backwards by an amount $z = (T_c - B_p)/2$.

Corollary 8.1. *The special case where $T_c > T_c^{\mathrm{EU}}$, $l = 1$, and $BR_p(B_p + \Delta B_p) = T_p$ is dominated by a situation with $l = 2$.* □

Lemma 8.4. *For $T_c > T_c^{\mathrm{EU}}$ and $l = 1$, we can construct a situation with a relative phasing ϕ_{R} and $l = 2$ such that at most the same amount of budget margin becomes available in T_c.*

Proof. Let the amount of budget margin that becomes available in the T_c-interval with a relative phasing φ_{R} be equal to B'. Because $T_c > T_c^{\mathrm{EU}}$, $B' > 0$. Moreover, because $l = 1$, $B' \le \Delta B_p$. Hence, we have three different situations to consider depending on the relative phasings φ_{R} on the one hand and the occupied time $O_p(B_p)$ and the response time $R_p(B_p + \Delta B_p)$ of the consumption of the normal budget B_p and the total budget $B_p + \Delta B_p$, respectively, in the period T_p on the other hand:

1. $\varphi_{\mathrm{R}} \le O_p(B_p)$;
2. $\varphi_{\mathrm{R}} > O_p(B_p)$ and $\varphi_{\mathrm{R}} + T_c \ge R_p(B_p + \Delta B_p)$;
3. $\varphi_{\mathrm{R}} > O_p(B_p)$ and $\varphi_{\mathrm{R}} + T_c < R_p(B_p + \Delta B_p)$.

For all three cases, we will construct a situation with a relative phasing ϕ such that $l = 2$ and a worst-case available budget margin that is at most equal to B'.

Case 1 is illustrated in Figure 8.4. Now assume a worst-case advancement of AGB_p in the current period T_p. Such an advancement will yield an amount of available budget margin in the interval T_c that is at most equal to B', because $B' = A_p(\varphi_{\mathrm{R}} + T_c) - B_p \ge WA_p(\varphi_{\mathrm{R}} + T_c) - B_p$. Moreover, assume a best-case advancement of AGB_p in the previous period T_p. The trivial case where $BR_p(B_p + \Delta B_p) = T_p$ is covered by Corollary 8.1. Let $BR_p(B_p + \Delta B_p) < T_p$. By moving the interval T_c backwards such that the relative phasing ϕ with respect to that previous period T_p equals $BR_p(B_p + \Delta B_p)$, the amount of available budget margin of the current period T_p remains at most the same, and the amount in the previous period T_p is zero. Hence, for case 1 a phasing $\phi = BR_p(B_p + \Delta B_p)$ in a

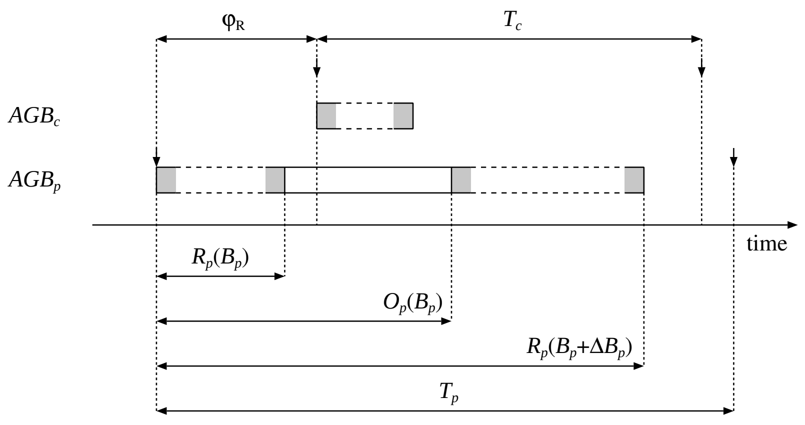


Figure 8.4. A situation with $l = 1$, and $\varphi_R \leq O_p(B_p)$.

situation with a worst-case provision yields at most B'. For this phasing ϕ, we have $l = 2$.

The reasoning for case 2 is similar. We first assume a best-case advancement of AGB_p in the current period T_p. This keeps the worst-case available budget margin at most the same, i.e. $B' = B_p + \Delta B_p - A_p(\varphi_R) \geq B_p + \Delta B_p - BA_p(\varphi_R)$. In addition, we assume a worst-case advancement in the next period, and finally move the interval T_c forward such that the relative phasing ϕ' of the next activation of T_c becomes equal to $WO_p(B_p)$. For this phasing, we again have $l = 2$.

Case 3 is slightly different from the other two cases. We first observe that

$$T_c \leq WO_p(B') \leq WO_p(B_p + B').$$

This is illustrated in Figure 8.5. The worst-case advancement in a T_c interval is

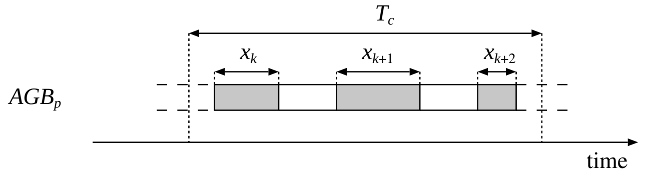


Figure 8.5. A situation with $l = 1$, i.e. $\varphi_R + T_c < T_p$, and a budget margin provision $B' = x_k + x_{k+1} + x_{k+2}$.

therefore at most $B_p + B'$, i.e.

$$WA_p(T_c) \leq WA_p(WO_p(B_p + B')) = B_p + B'.$$

Hence, a worst-case advancement in the current period T_p combined with a relative phasing $\phi = 0$ yields at most an amount of budget margin B', i.e.

$$\Delta B_c(0, T_c) = WA_p(T_c) - B_p \leq B'.$$

In addition, we assume a best-case advancement of AGB_p in the previous period, and move the interval T_c backwards such that ϕ becomes $BR_p(B_p + \Delta B_p)$. During the move, the available budget margin remains at most the same, and for this phasing we again have $l = 2$. □

8.3.3 Periodicity

The proof of Lemma 8.1 is based on a construction argument using two successive releases of AGB_p. Using a similar argument for $k+2$ successive releases of AGB_p with $k \in \mathbb{N}^+$, we derive $\Delta B_c(T_c^{\mathrm{EU}} + kT_p) = k\Delta B_p$. The next lemma concerning the periodicity of $\Delta B_c(T_c)$ provides a generalization of the latter equation for arbitrary T_c larger than or equal to T_c^{EU}.

Lemma 8.5. *For $T_c \geq T_c^{\mathrm{EU}}$, the shape of $\Delta B_c(T_c)$ is periodic with period T_p, i.e.*

$$\Delta B_c(T_c + kT_p) = \Delta B_c(T_c) + k\Delta B_p \text{ for } T_c \geq T_c^{\mathrm{EU}} \wedge k \in \mathbb{N}^+. \tag{8.11}$$

Proof. As mentioned above, the worst-case provision for a specific phasing φ_{R} is characterized by the first and the last overlapping interval, irrespective of the number of overlapping intervals. This forms the basis of our proof.

Consider a specific relative phasing φ_{R} and let $T_c \geq T_c^{\mathrm{EU}}$. Let l denote the number of overlapping intervals for φ_{R} and T_c, and φ'_{R} the relative phasing of the next release of AGB_c. Remember that we may restrict ourselves to the case where $l \geq 2$. Now assume the same relative phasing φ_{R} for a period T'_c where $T'_c = T_c + kT_p$ with $k \in \mathbb{N}^+$. Below, we will first prove that the number of overlapping intervals l' for φ_{R} and T'_c is equal to $l + k$, and that the relative phasing φ''_{R} of the next release is identical to φ'_{R}. We now prove $l' = l + k$, and $\varphi''_{\mathrm{R}} = \varphi'_{\mathrm{R}}$.

$$\begin{aligned} l' &= \lceil (\varphi_{\mathrm{R}} + T'_c)/T_p \rceil = \lceil (\varphi_{\mathrm{R}} + T_c + kT_p)/T_p \rceil \\ &= \lceil (\varphi_{\mathrm{R}} + T_c)/T_p \rceil + k = l + k \\ \varphi''_{\mathrm{R}} &= (\varphi_{\mathrm{R}} + T'_c) \bmod T_p = (\varphi_{\mathrm{R}} + T_c + kT_p) \bmod T_p \\ &= (\varphi_{\mathrm{R}} + T_c) \bmod T_p = \varphi'_{\mathrm{R}} \end{aligned}$$

Hence, the worst-case situations for T_c and T'_c only differ in the number of overlapping intervals, being k. Each of these k intervals contributes an additional ΔB_p to $\Delta B_c(T_c + kT_p)$ compared to $\Delta B_c(T_c)$. Because this argument holds for an arbitrary relative phasing φ_{R}, this proves the lemma. □

We will now summarize our findings and arrive at initial bounds for budget margin provision. For values of T_c in the interval $[0, T_c^{\mathrm{EU}}]$, no budget can be conditionally guaranteed under arbitrary phasings and on a strictly periodic basis, i.e. $\Delta B_c(T_c) = 0$. For $T_c \geq T_c^{\mathrm{EU}} + T_p$, we can apply Lemma 8.5, i.e. $\Delta B_c(T_c) = \Delta B_c(T_c - T_p) + \Delta B_p$, and possibly we can apply this lemma repeatedly. For the remaining interval, i.e. $T_c \in (T_c^{\mathrm{EU}}, T_c^{\mathrm{EU}} + T_p)$, $\Delta B_c(T_c) \in (0, \Delta B_p]$. Because the gradient of $\Delta B_c(T_c)$ is at most one, this can be refined to

$$T_c \in (T_c^{\mathrm{EU}}, T_c^{\mathrm{EU}} + T_p) \Rightarrow \Delta B_{c,\mathrm{lwb}}(T_c) \leq \Delta B_c(T_c) \leq \Delta B_{c,\mathrm{upb}}(T_c) \tag{8.12}$$

where

$$\Delta B_{c,\mathrm{lwb}}(T_c) = \max\{0, T_c + \Delta B_p - T_c^{\mathrm{EU}} - T_p\} \tag{8.13}$$

$$\Delta B_{c,\mathrm{upb}}(T_c) = \min\{\Delta B_p, T_c - T_c^{\mathrm{EU}}\}. \tag{8.14}$$

Figure 8.6 illustrates these initial bounds for budget margin provision for our example. In the next subsections, we will tighten these initial bounds.

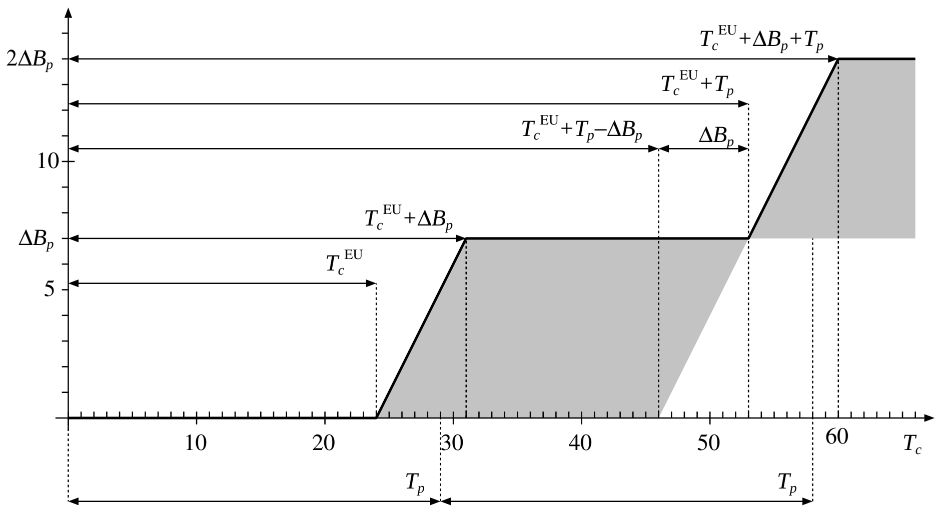


Figure 8.6. Initial bounds for budget margin provision. The grey areas denote where exact values for $\Delta B_c(T_c)$ can be found.

8.3.4 Exact lower bound

By means of (8.6) on page 135, we defined the exact upper bound $T_c^{\mathrm{EU}}(x)$. Similarly, we can define an *exact lower bound* $T_c^{\mathrm{EL}}(x)$ as the minimum T_c for which an amount x can be conditionally guaranteed on a strictly periodic basis under arbitrary phasing, where $x \in \mathbb{R}$ and $x \geq 0$, i.e.

$$T_c^{\mathrm{EL}}(x) = \inf\{T_c \mid \Delta B_c(T_c) = x\}. \tag{8.15}$$

Obviously, $T_c^{\mathrm{EL}}(0) = 0$. Moreover, $T_c^{\mathrm{EL}}(x) \leq T_c^{\mathrm{EU}}(x)$ by definition. Because we are primarily interested in the exact lower bound for which the entire budget margin

ΔB_p can be guaranteed conditionally, we will use T_c^{EL} as a shorthand for $T_c^{\mathrm{EL}}(\Delta B_p)$. As mentioned before, $\Delta B_c^{\mathrm{avg}}$ (7.1) on page 118 is an upper bound on the amount that can be conditionally guaranteed based on the budget margin on a periodic basis. Because $\Delta B_c^{\mathrm{avg}} = \Delta B_p$ for $T_c = T_p$, we conclude that $T_p \leq T_c^{\mathrm{EL}}$.

Based on a construction argument, we proved in Lemma 8.1 that the exact upper bound T_c^{EU} is given by (8.7). Using that equation, we can easily determine the value of T_c^{EU}. Unlike T_c^{EU}, it is not clear how to derive a similar simple algorithm to determine T_c^{EL}, and such an algorithm may not even exist.

In Section 8.4, we will derive that for our example $T_c^{\mathrm{EL}} = 35$, which is considerably smaller than $T_c^{\mathrm{EU}} + T_p = 24 + 29 = 53$.

8.3.5 Pessimistic upper bound and pessimistic lower bound

Although we have no simple algorithm to determine T_c^{EL}, we can easily determine a *pessimistic lower bound*[1] $T_c^{\mathrm{PL}} \geq T_c^{\mathrm{EL}}$. This value is pessimistic in the sense that there may (but need not) exist values T_c smaller than T_c^{PL} for which the entire budget margin ΔB_p can be provided. Moreover, T_c^{PL} is typically smaller than $T_c^{\mathrm{EU}} + T_p$, although it may also be larger for specific cases.

For ease of presentation, we define a *pessimistic upper bound* $T_c^{\mathrm{PU}} \geq T_c^{\mathrm{EU}}$. This value is pessimistic in the sense that the amount of budget margin that can be provided may (but need not) be larger than zero, i.e. $\Delta B_c(T_c^{\mathrm{PU}}) \geq 0$.

To determine T_c^{PU} and T_c^{PL}, we make pessimistic assumptions about the provision of the budget margin ΔB_p, namely we assume that the budget margin ΔB_p becomes available without preemptions. Moreover, we assume that ΔB_p becomes available as early as possible for the best-case advancement (i.e. immediately after the best-case occupied time of the normal budget $BO_p(B_p)$) and as late as possible for the worst-case advancement (i.e. just before the worst-case response time of the combined budgets $WR_p(B_p + \Delta B_p)$). The dotted lines in Figure 8.7 illustrate the assumptions.

From Figure 8.7, we derive:

$$T_c^{\mathrm{PU}} = T_p + WR_p(B_p + \Delta B_p) - BO_p(B_p) - 2\Delta B_p \qquad (8.16)$$

$$T_c^{\mathrm{PL}} = T_c^{\mathrm{PU}} + \Delta B_p. \qquad (8.17)$$

For our example, we find $T_c^{\mathrm{PU}} = 29$ and $T_c^{\mathrm{PL}} = 36$. Because $T_c^{\mathrm{EU}} = 24$ and $T_c^{\mathrm{EL}} = 35$, both bounds are actually pessimistic, i.e. $T_c^{\mathrm{EU}} < T_c^{\mathrm{PU}}$ and $T_c^{\mathrm{EL}} < T_c^{\mathrm{PL}}$.

For the highest priority AGB, the response times and occupied times are equal

[1]Be aware that the *pessimistic lower bound* is *at least* the *exact lower bound*. The usage of the term *lower bound* may therefore be confusing.

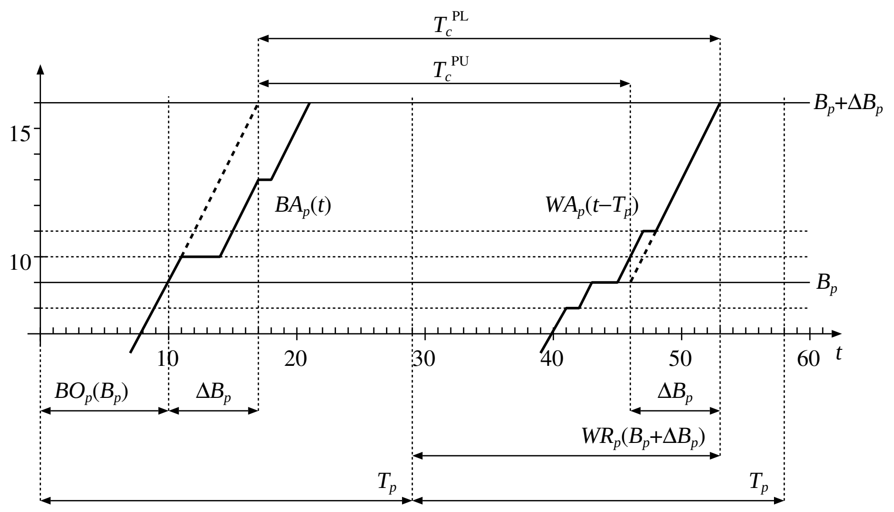


Figure 8.7. Pessimistic assumptions about the availability of ΔB_p indicated by the dotted lines.

to the budget, and we therefore find that T_c^{EU} and T_c^{PU} are equal, i.e.

$$\begin{aligned} T_c^{\mathrm{EU}} &= \{(8.7)\}\ T_p + WO_p(B_p) - BR_p(B_p+\Delta B_p) \\ &= \{AGB_p \text{ has highest priority}\}\ T_p + B_p - (B_p+\Delta B_p) \\ &= T_p - \Delta B_p \end{aligned}$$

and

$$\begin{aligned} T_c^{\mathrm{PU}} &= \{(8.16)\}\ T_p + WR_p(B_p+\Delta B_p) - BO_p(B_p) - 2\Delta B_p \\ &= \{AGB_p \text{ has highest priority}\}\ T_p + B_p + \Delta B_p - B_p - 2\Delta B_p \\ &= T_p - \Delta B_p. \end{aligned}$$

As mentioned above, $\Delta B_c(T_c)$ increases at most with a gradient of one, hence $T_c^{\mathrm{EU}} + \Delta B_p \leq T_c^{\mathrm{EL}}$. For the highest priority AGB we therefore find

$$\begin{aligned} T_c^{\mathrm{EU}} + \Delta B_p &\leq T_c^{\mathrm{EL}} \\ &\leq T_c^{\mathrm{PL}} \\ &= \{(8.17)\}\ T_c^{\mathrm{PU}} + \Delta B_p \\ &= \{AGB_p \text{ has highest priority}\}\ T_c^{\mathrm{EU}} + \Delta B_p, \end{aligned}$$

i.e. $T_c^{\mathrm{EL}} = T_c^{\mathrm{PL}} = T_c^{\mathrm{EU}} + \Delta B_p$.

The following lemma describes under which conditions $\Delta B_c(T_c^{\mathrm{PL}}) = \Delta B_p$. For our example, $T_c^{\mathrm{PL}} = 36 < T_c^{\mathrm{EU}} + T_p = 53$, hence we can conclude $\Delta B_c(T_c) = \Delta B_p$ for $T_c \in [36, 53]$.

Lemma 8.6. *The following relation holds for T_c^{PL}.*

$$T_c^{\mathrm{PL}} \leq T_c^{\mathrm{EU}} + T_p \Leftrightarrow \Delta B_c(T_c^{\mathrm{PL}}) = \Delta B_p \tag{8.18}$$

Proof.

$\{\Rightarrow\}$ $\Delta B_c(T_c)$ is strictly non-decreasing. Hence, from $T_c^{\mathrm{PL}} \leq T_c^{\mathrm{EU}} + T_p$ we derive $\Delta B_c(T_c^{\mathrm{PL}}) \leq \Delta B_c(T_c^{\mathrm{EU}} + T_p)$. The left-hand side of this latter relation is larger than or equal to ΔB_p by definition, i.e. $\Delta B_p \leq \Delta B_c(T_c^{\mathrm{PL}})$ and the right-hand side is equal to ΔB_p. Hence, the implication holds.

$\{\Leftarrow\}$ $T_c^{\mathrm{EU}} + T_p$ is the largest T_c for which $\Delta B_c(T_c) = \Delta B_p$. Hence, from $\Delta B_c(T_c^{\mathrm{PL}}) = \Delta B_p$ we derive $T_c^{\mathrm{PL}} \leq T_c^{\mathrm{EU}} + T_p$. □

Using T_c^{PL}, we will now present improved bounds for $\Delta B_c(T_c)$. When $T_c^{\mathrm{PL}} \geq T_c^{\mathrm{EU}} + T_p$, the initial bounds given by (8.12) on page 140 can not be improved. For $T_c^{\mathrm{PL}} < T_c^{\mathrm{EU}} + T_p$, we consider two intervals:

$$T_c \in (T_c^{\mathrm{EU}}, T_c^{\mathrm{PL}}) \Rightarrow \Delta B_{c,\mathrm{lwb}'}(T_c) \leq \Delta B_c(T_c) \leq \Delta B_{c,\mathrm{upb}'}(T_c) \tag{8.19}$$

where

$$\Delta B_{c,\mathrm{lwb}'}(T_c) = \max\{0, T_c + \Delta B_p - T_c^{\mathrm{PL}}\} \tag{8.20}$$

$$\Delta B_{c,\mathrm{upb}'}(T_c) = \min\{\Delta B_p, T_c - T_c^{\mathrm{EU}}\} \tag{8.21}$$

and

$$T_c \in [T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p) \Rightarrow \Delta B_c(T_c) = \Delta B_p. \tag{8.22}$$

For our example, this is illustrated in Figure 8.8. Exact values for the worst-case available budget margin for $T_c \in (T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$ are determined in the next section.

8.4 Exact amount of worst-case available budget margin

In this section, we will determine exact values for the worst-case available budget margin for the remaining interval $T_c \in (T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$. To this end, we first introduce the notions of best-case and worst-case CGB contribution functions, being the contributions of the best-case and worst-case advancements of the budget margin of AGB_p to ΔB_c.

8.4.1 CGB contribution functions

Consider a period $T_c = 31$ and a relative phasing $\varphi_{\mathrm{R}} = 18$ for our example. According to (8.9), the number l of overlapping intervals of length T_p is two. Figure 8.9 illustrates the worst-case available $\Delta B_c(\varphi_{\mathrm{R}}, T_c)$. For this phasing, only parts of the best-case and worst-case advancement of the budget margin ΔB_p become available. Those parts are indicated by the vertical arrows and denoted by

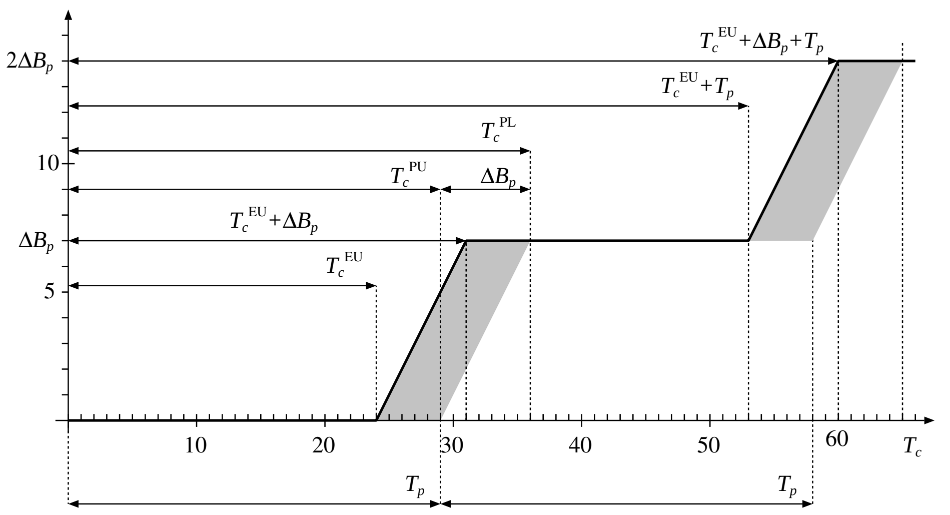


Figure 8.8. Improved bounds for budget margin provision. The grey areas denote where exact values for $\Delta B_c(T_c)$ can be found.

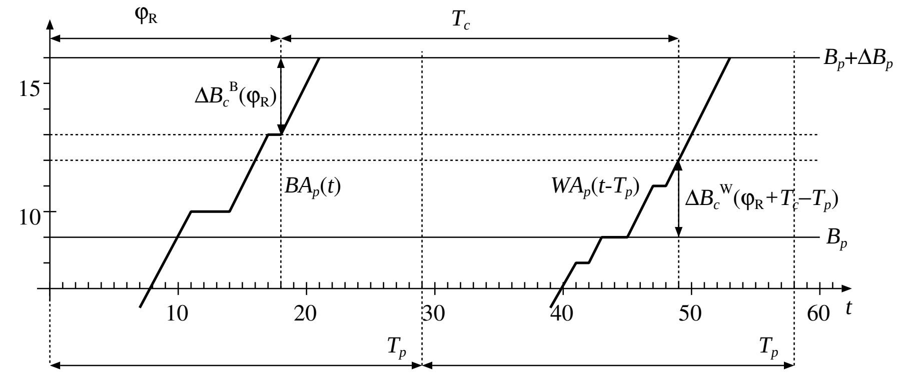


Figure 8.9. Available ΔB_c for $T_c = 31$ and $\varphi_R = 18$ for our example.

means of $\Delta B_c^B(\varphi_R)$ and $\Delta B_c^W(\varphi_R + T_c - T_p)$, respectively. The term $\Delta B_c^B(\varphi_R)$ represents the contribution from the best-case advancement to $\Delta B_c(\varphi_R, T_c)$ and the term $\Delta B_c^W(\varphi_R + T_c - T_p)$ represents the contribution from the worst-case advancement to $\Delta B_c(\varphi_R, T_c)$. In the remainder, ΔB_c^B and ΔB_c^W will be termed the *best-case CGB contribution function* and the *worst-case CGB contribution function*, respectively. These functions are given by the following equations.

$$\Delta B_c^B(t) = \max\{0, \min\{\Delta B_p, B_p + \Delta B_p - BA_p(t)\}\} \quad (8.23)$$

$$\Delta B_c^W(t) = \max\{0, \min\{\Delta B_p, WA_p(t) - B_p\}\} \quad (8.24)$$

From these equations, we immediately derive that $\Delta B_c^{\mathrm{B}}(t)$ is a monotonic non-increasing function of t, and $\Delta B_c^{\mathrm{W}}(t)$ is a monotonic non-decreasing function of t. Figure 8.10 depicts $\Delta B_c^{\mathrm{B}}(t)$ and $\Delta B_c^{\mathrm{W}}(t)$ for our example. Both CGB contribution

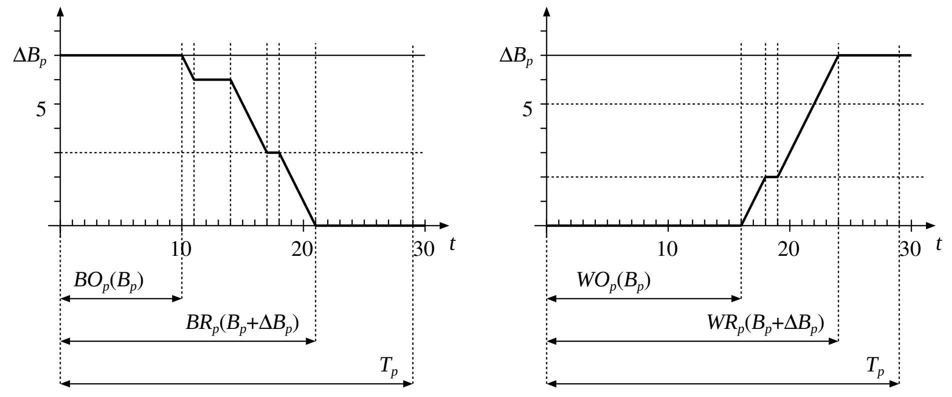


Figure 8.10. Best-case CGB contribution function ΔB_c^{B} and worst-case CGB contribution function ΔB_c^{W} for our example.

functions bend at least twice (for $\Delta B_p > 0$) and an even number of times. $\Delta B_c^{\mathrm{B}}(t)$ bends from horizontal to diagonal for a so-called *BO*-bending point and from diagonal to horizontal for a so-called *BR*-bending point. Similarly, $\Delta B_c^{\mathrm{W}}(t)$ bends from horizontal to diagonal for a so-called *WO*-bending point and from diagonal to horizontal for a so-called *WR*-bending point. The worst-case available ΔB_c is now given by the following equation for $l = 2$.

$$\Delta B_c(\varphi_{\mathrm{R}}, T_p) = \Delta B_c^{\mathrm{B}}(\varphi_{\mathrm{R}}) + \Delta B_c^{\mathrm{W}}(\varphi_{\mathrm{R}} + T_c - T_p) \tag{8.25}$$

The amount of budget margin that becomes available during the activation with a relative phasing $\varphi_{\mathrm{R}} = 18$ is for our example given by

$$\Delta B_c(18, 31) = \Delta B_c^{\mathrm{B}}(18) + \Delta B_c^{\mathrm{W}}(18 + 31 - 29) = 3 + 3 = 6.$$

Figure 8.11 shows the available ΔB_c as a function of the relative phasing φ_{R} for $T_c = 31$ using best-case and worst-case contribution functions ΔB_c^{B} and ΔB_c^{W}. From Figure 8.11, we derive that the available ΔB_c for a period T_c equal to 31 has a minimum for a relative phasing φ_{R} equal to 17. Hence, $\Delta B_c(31) = \Delta B_c(17, 31) = 5$.

8.4.2 Cognac-glass algorithm

In the previous section, we illustrated by means of Figure 8.11 how ΔB_c can be determined for a particular period. In that figure, we moved the worst-case CGB contribution function ΔB_c^{W} an amount $T_c - T_p$ to the *left*. In this section, we will use an alternative drawing that can be used for arbitrary periods. In that drawing, the function ΔB_c^{W} is moved a fixed amount T_p to the *right*; see Figure 8.12. In order

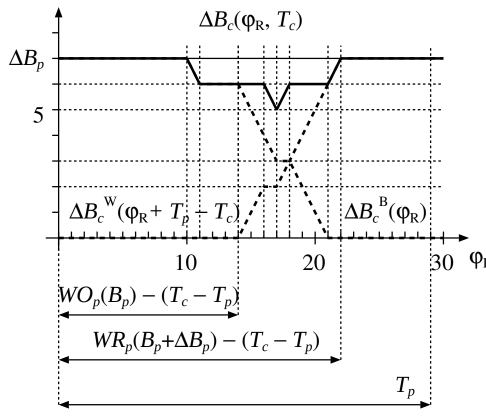


Figure 8.11. Construction of $\Delta B_c(\varphi_R, T_c)$ as a function of φ_R for $T_c = 31$ using CGB contribution functions. $\Delta B_c(\varphi_R, T_c)$ is minimal for $\varphi_R = 17$.

to determine the amount of budget $\Delta B_c(T_c)$ that can be conditionally guaranteed for a particular period T_c in the interval $(T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$, we apply a so-called *cognac-glass algorithm* (CGA). This algorithm takes a particular period T_c and determines $\Delta B_c(T_c)$ by taking the minimum of $\Delta B_c(\varphi_R, T_c)$ for the values of the relative phasing φ_R. The term 'cognac-glass algorithm' originates from the observation that the slanted lines may be viewed as the shape of a glass, and the minimal sum of the heights of the cognac at both sides of the glass when tilting the glass determines the budget for a specific period[2]. We will first show that we only need to consider a particular range of values of φ_R to determine $\Delta B_c(T_c)$. We subsequently present the result of applying the algorithm, and finally describe the complexity of the algorithm in terms of the number of times we need to evaluate $\Delta B_c(\varphi_R, T_c)$ for our example.

Lemma 8.7. *For $T_c \in (T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$, a minimal (i.e. worst-case) value for $\Delta B_c(\varphi_R, T_c)$ is found for $\varphi_R \in [WO_p(B_p) + T_p - T_c, BR_p(B_p + \Delta B_p)]$.*

Proof. We first prove that the interval is non-empty. We subsequently prove the lemma.

From $T_c > T_c^{\mathrm{EU}}$ and (8.7), we get

$$\begin{aligned} & T_c > WO_p(B_p) + T_p - BR_p(B_p + \Delta B_p) \\ \Leftrightarrow \quad & BR_p(B_p + \Delta B_p) > WO_p(B_p) + T_p - T_c. \end{aligned}$$

[2]Strictly speaking, the term 'cognac-glass algorithm' is suggestive, because the projected length of the surface of the cognac (i.e. T_c) in the glass is kept constant, rather than the amount of cognac below the surface.

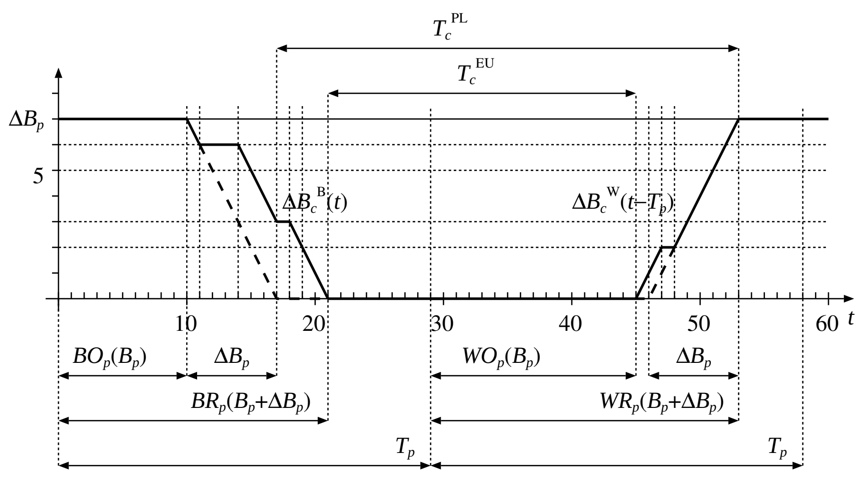


Figure 8.12. An alternative drawing based on CGB contribution functions. Pessimistic assumptions about the availability of ΔB_p are indicated by the dashed lines.

Hence, the interval is non-empty.

The earliest moment after the release of the AGB_p that the entire budget margin is completed is given by $BR_p(B_p + \Delta B_p)$. Similarly, the latest moment after the release of the AGB_p that the budget margin becomes available is given by $WO_p(B_p)$. Let the start of the T_c-interval $\varphi_R > BR_p(B_p + \Delta B_p)$. When φ_R is decreased to $BR_p(B_p + \Delta B_p)$, the total amount of budget margin that becomes available from the overlapping intervals either remains the same or becomes less. Hence, when φ_R is decreased to $BR_p(B_p + \Delta B_p)$, $\Delta B_c(\varphi_R, T_c)$ is strictly non-increasing. Similarly, let the end of the T_c-interval $\varphi_R + T_c < WO_p(B_p) + T_p$, i.e. $\varphi_R < WO_p(B_p) + T_p - T_c$. When φ_R is increased to $WO_p(B_p) + T_p - T_c$, the total amount of budget margin that becomes available from the overlapping intervals is also strictly non-increasing. A minimal (i.e. worst-case) value for ΔB_c under arbitrary phasings is therefore found when $\varphi_R \in [WO_p(B_p) + T_p - T_c, BR_p(B_p + \Delta B_p)]$. □

Because we consider values of T_c in the interval $(T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$, we are only interested in values of φ_R for which $\Delta B_c(\varphi_R, T_c) < \Delta B_p$. Stated in terms of our metaphor, we don't want to spoil cognac across the rim when tilting the glass too far. Hence, if $\varphi_R + T_c > WR_p(B_p + \Delta B_p) + T_p$ for $\varphi_R = BR_p(B_p + \Delta B_p)$, we can decrease the upper bound of the interval for φ_R to $WR_p(B_p + \Delta B_p) + T_p - T_c$. Similarly, if $\varphi_R < BO_p(B_p)$ for $\varphi_R + T_c = WO_p(B_p) + T_p$, we can increase the lower bound of the interval for φ_R to $BO_p(B_p)$. In summary, for $T_c \in (T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$ we only need to consider values of φ_R in the range

$[\varphi_{\mathrm{R,lwb}}(T_c), \varphi_{\mathrm{R,upb}}(T_c)]$, where

$$\varphi_{\mathrm{R,lwb}}(T_c) = \max\{BO_p(B_p), WO_p(B_p) + T_p - T_c\} \quad (8.26)$$

$$\varphi_{\mathrm{R,upb}}(T_c) = \min\{BR_p(B_p + \Delta B_p), WR_p(B_p + \Delta B_p) + T_p - T_c\} \quad (8.27)$$

under the condition that we take the minimum of ΔB_p and the result found, i.e.

$$\Delta B_c(T_c) = \min\{\Delta B_p, \min\{\Delta B_c(\varphi_{\mathrm{R}}, T_c) | \varphi_{\mathrm{R}} \in [\varphi_{\mathrm{R,lwb}}(T_c), \varphi_{\mathrm{R,upb}}(T_c)]\}\}. \quad (8.28)$$

Figure 8.13 depicts the exact worst-case available $\Delta B_c(T_c)$ under arbitrary phasing for our example. From this figure, we derive that $T_c^{\mathrm{EL}} = 35$.

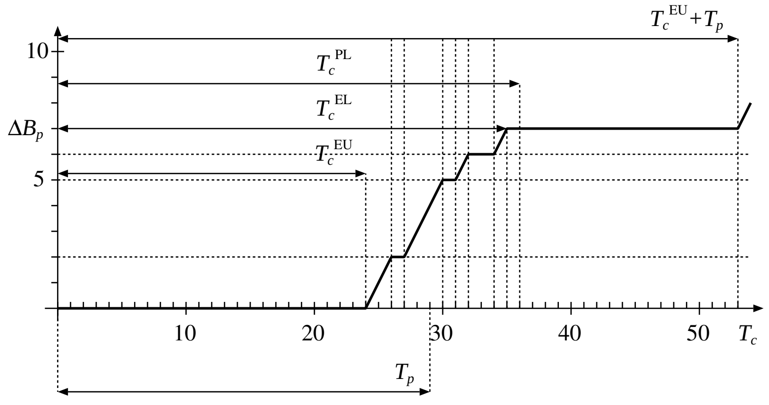


Figure 8.13. Exact worst-case available $\Delta B_c(T_c)$.

We will now briefly describe the complexity of determining the graph shown in Figure 8.13 for this particular example. Because all budget characteristics are natural numbers in this example, we may restrict ourselves to values of T_c as well as φ_{R} belonging to the natural numbers. We start with determining $\Delta B_c(T_c)$ for $T_c = T_c^{\mathrm{EU}} + 1$. We continue with increasing values for T_c till we either encounter $\Delta B_c(T_c) = \Delta B_p$, hence $T_c = T_c^{\mathrm{EL}}$, or $T_c = \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\} - 1$. For a particular period T_c, we determine $\Delta B_c(\varphi_{\mathrm{R}}, T_c)$ using (8.28). For our example, $T_c^{\mathrm{EL}} = T_c^{\mathrm{PL}} - 1 < T_c^{\mathrm{EU}} + T_p$. The total number of times η that (8.25) needs to be evaluated is therefore equal to

$$\begin{aligned}\eta &= \sum_{T_c=T_c^{\mathrm{EU}}+1}^{T_c^{\mathrm{PL}}-1} (\varphi_{\mathrm{R,upb}}(T_c) - \varphi_{\mathrm{R,lwb}}(T_c) + 1) \\ &= \sum_{T_c=T_c^{\mathrm{EU}}+1}^{T_c^{\mathrm{PL}}-1} (\min\{BR_p(B_p + \Delta B_p), WR_p(B_p + \Delta B_p) + T_p - T_c\} \\ &\qquad - \max\{BO_p(B_p), WO_p(B_p) + T_p - T_c\} + 1).\end{aligned}$$

For our example, this yields

$$\begin{aligned}
\eta &= \sum_{T_c=25}^{35} (\min\{21, 24+29-T_c\} - \max\{10, 16+29-T_c\}+1) \\
&= \sum_{T_c=25}^{35} (\min\{21, 53-T_c\} - (45-T_c)+1) \\
&= \{i = T_c - 23\} \sum_{i=2}^{12} (\min\{21, 30-i\} - (21-i)) \\
&= \sum_{i=2}^{9} i + \sum_{i=10}^{12} 9 = 44 + 27 = 71.
\end{aligned}$$

8.5 Efficient calculation of conditionally guaranteed budgets

This section presents an efficient algorithm to calculate CGBs, which uses the ranges for T_c identified in Section 8.3. We first prove that for $T_c \in (T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$ we only need to consider a restricted set of values for φ_{R} in order to determine $\Delta B_c(T_c)$. We next present our algorithm to determine $\Delta B_c(T_c)$.

8.5.1 Dominating values for φ_{R}

According to Lemma 8.7 and (8.28), we only need to consider values of φ_{R} in a restricted domain in order to determine the minimum of $\Delta B_c(\varphi_{\mathrm{R}}, T_c)$ for given T_c from the interval $(T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$. Considering Figure 8.11, we observe that $\Delta B_c(\varphi_{\mathrm{R}}, T_c)$ has a minimum for $\varphi_{\mathrm{R}} = 17$. This value for φ_{R} corresponds with a *BR*-bending point of the CGB contribution function $\Delta B_c^{\mathrm{B}}(\varphi_{\mathrm{R}})$, and a *WO*-bending point of $\Delta B_c^{\mathrm{W}}(\varphi_{\mathrm{R}} + T_c - T_p)$. The next lemma states that given an interval $[w, z]$, the boundaries w and z together with the *BR*-bending points of $\Delta B_c^{\mathrm{B}}(\varphi)$ in (w, z) are dominating values for φ_{R} to determine the minimum of $\Delta B_c(\varphi_{\mathrm{R}}, T_c)$ in $[w, z]$. A similar lemma is subsequently presented using *WO*-bending points. Both Lemmas are illustrated by means of an example.

Lemma 8.8. *For $T_c \in (T_c^{\mathrm{EU}}, \min\{T_c^{\mathrm{PL}}, T_c^{\mathrm{EU}} + T_p\})$, a minimal (i.e. worst-case) value for $\Delta B_c(\varphi_{\mathrm{R}}, T_c)$ for given T_c and φ_{R} in the range $[w, z]$ is found by considering the boundaries w and z of the interval and all BR-bending points of ΔB_c^{B} in (w, z).*

Proof. We take (8.25) as a starting point, and base the proof on the shape of the graphs of the CGB contribution functions.

The gradient of a line segment of $\Delta B_c^{\mathrm{B}}(\varphi_{\mathrm{R}})$ is either 0 or -1. Similarly, the gradient of a line segment of $\Delta B_c^{\mathrm{W}}(\varphi_{\mathrm{R}})$ is either 0 or $+1$. Let $f(\varphi_{\mathrm{R}}) = \Delta B_c(\varphi_{\mathrm{R}}, T_c)$.

The function $f(\varphi_\mathrm{R})$ therefore has a gradient of -1, 0, or $+1$. Apart from the boundaries w and z, it is sufficient to consider the horizontal line segments in (w,z), that are preceded by a line segment with a gradient -1 and succeeded by a line segment with a gradient $+1$. For each of those horizontal line segments, it suffices to consider only a single, arbitrary, point. As illustrated by Figure 8.11, such a horizontal line segment may specialize to a single point. We will now prove that by considering all *BR*-bending points in (w,z), all horizontal line segments are covered.

Consider a horizontal line segment $[x,y]$. Just before that line segment, the gradient of the graph is -1, i.e.

$$f'(x^-) = -1 \Rightarrow \Delta B_c^{\mathrm{B}\prime}(x^-) = -1.$$

Just after that line segment, the gradient is $+1$, i.e.

$$f'(y^+) = +1 \Rightarrow \Delta B_c^{\mathrm{B}\prime}(y^+) = 0.$$

Hence, somewhere in the interval $[x,y]$ we can find a value for which ΔB_c^B bends from from -1 to 0, and for that value we have a *BR*-bending point. We therefore conclude that $[x,y]$ contains a *BR*-bending point. By considering all *BR*-bending points in (w,z), we therefore cover all horizontal line segments in (w,z). The set containing the *BR*-bending points in (w,z) together with the boundaries w and z therefore dominates $[w,z]$ when determining the minimum of $f(\varphi_\mathrm{R})$ in $[w,z]$. □

We may formulate a similar lemma for *WO*-bending points.

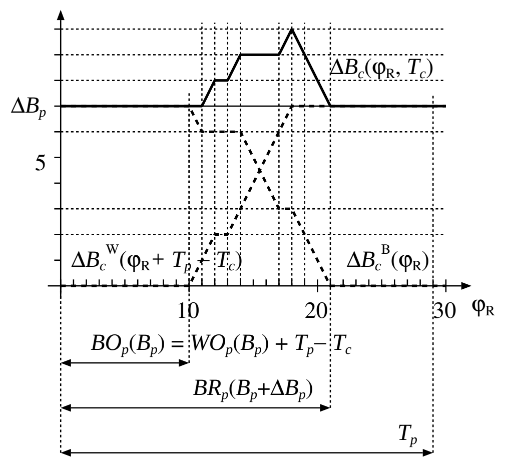
Lemma 8.9. *For $T_c \in (T_c^\mathrm{EU}, \min\{T_c^\mathrm{PL}, T_c^\mathrm{EU} + T_p\})$, a minimal (i.e. worst-case) value for $\Delta B_c(\varphi_\mathrm{R}, T_c)$ for given T_c and φ_R in the range $[w,z]$ is found by considering the boundaries w and z of the interval and all WO-bending points of ΔB_c^W in (w,z).*

Proof. Similar to that of Lemma 8.8. □

We will now illustrate these lemmas by means of an example. Figure 8.14 shows $\Delta B_c(\varphi_\mathrm{R}, T_c)$ as a function of φ_R for $T_c = T_c^\mathrm{EL} = 35$. Using (8.28), we only need to consider values of φ_R in the interval $[\varphi_{\mathrm{R,lwb}}(35), \varphi_{\mathrm{R,upb}}(35)] = [10,18]$.

Note that the graph does not have any horizontal line-segment in the interval $[10,18]$ that is both preceded by a line segment with gradient -1 and succeeded by a line segment with gradient $+1$. Hence, although the *BR*-bending points of $\Delta B_c^\mathrm{B}(\varphi_\mathrm{R})$ in $[10,18]$ for $\varphi_\mathrm{R} = 11$ and $\varphi_\mathrm{R} = 17$ cover horizontal line-segments, they do not correspond with a minimum for $\Delta B_c^\mathrm{B}(\varphi_\mathrm{R})$. A similar observation holds for the *WO*-bending point for $\varphi_\mathrm{R} = 13$.

In order to determine the minimum of $\Delta B_c(\varphi, 35)$ using Lemma 8.8, we only need to consider four values for φ_R, i.e. values in $\Phi^\mathrm{B} = \{10,11,17,18\}$. Based



ΔB_p
5
$\Delta B_c(\varphi_R, T_c)$
$\Delta B_c^W(\varphi_R + T_p - T_c)$
$\Delta B_c^B(\varphi_R)$
10
20
30
φ_R
$BO_p(B_p) = WO_p(B_p) + T_p - T_c$
$BR_p(B_p + \Delta B_p)$
T_p

Figure 8.14. Construction of $\Delta B_c(\varphi_R, T_c)$ as a function of φ_R for $T_c = T_c^{EL}$ using CGB contribution functions for our example.

on Lemma 8.9, we find that we may also restrain ourselves to values in $\Phi^W = \{10, 13, 18\}$. We conclude this section with Figure 8.15, showing the complexity of the use of Lemma 8.8 for our example.

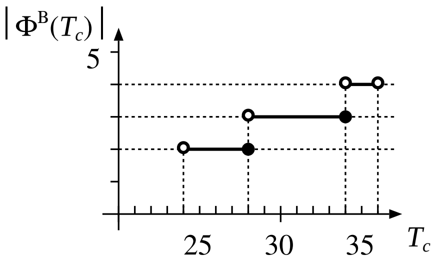


Figure 8.15. Number of times that (8.25) needs to be evaluated using Lemma 8.8 for our example for $T_c \in (T_c^{EU}, \min\{T_c^{PL}, T_c^{EU} + T_p\})$ (i.e. the cardinality of $\Phi^B(T_c)$).

8.5.2 Complete algorithm

The algorithm consists of three steps, based on the ranges of T_c as described below.

$$\begin{aligned}
T_c &\in [0, T_c^{EU}] && \Rightarrow \Delta B_c(T_c) = 0 \\
T_c &\in (T_c^{EU}, \min\{T_c^{PL}, T_c^{EU} + T_p\}) && \Rightarrow \Delta B_c(T_c) \in [\Delta B_{c,\text{lwb}'}(T_c), \Delta B_{c,\text{upb}'}(T_c)] \\
T_c &\in [T_c^{PL}, T_c^{EU} + T_p] && \Rightarrow \Delta B_c(T_c) = \Delta B_p \\
T_c &\in [T_c^{EU} + T_p, \infty) && \Rightarrow \Delta B_c(T_c) = \Delta B_c(T_c - T_p) + \Delta B_p
\end{aligned}$$

Step 1: Determine the range and optionally normalize T_c

We first determine T_c^{EU} and T_c^{PL} using (8.7) and (8.17), respectively. When T_c belongs to the fourth range, we determine the $k \in \mathbb{N}$ such that $T_c^{\mathrm{EU}} + kT_p \leq T_c < T_c^{\mathrm{EU}} + (k+1)T_p$, i.e. $k = \lfloor (T_c - T_c^{\mathrm{EU}})/T_p \rfloor$. Next, we determine the *normalized value* $T_c^{\mathrm{N}} = T_c - kT_p$ of T_c, i.e. $T_c^{\mathrm{EU}} \leq T_c^{\mathrm{N}} < T_c^{\mathrm{EU}} + T_p$.

Step 2: Determine ΔB_c for first and third range

If T_c belongs to the first or third range, we can determine $\Delta B_c(T_c)$ immediately from the formula, and we are done. If $T_c^{\mathrm{N}} = T_c^{\mathrm{EU}}$, we can find $\Delta B_c(T_c)$ by means of $\Delta B_c(T_c) = \Delta B_c(T_c^{\mathrm{N}}) + k\Delta B_p = k\Delta B_p$. Similarly, if T_c^{N} belongs to the third range, we can find $\Delta B_c(T_c)$ by means of $\Delta B_c(T_c) = \Delta B_c(T_c^{\mathrm{N}}) + k\Delta B_p = (k+1)\Delta B_p$. When T_c or T_c^{N} is in the second range, we determine ΔB_c using Lemma 8.8 and (8.28) in step 3. Note that for T_c in the second range, we may also use $T_c^{\mathrm{N}} = T_c$ and $k = 0$.

Step 3: Determine ΔB_c for second range

Let Φ^{B} be the set of phasings φ_{R} consisting of $\varphi_{\mathrm{R,lwb}}(T_c^{\mathrm{N}})$ (see (8.26)), $\varphi_{\mathrm{R,upb}}(T_c^{\mathrm{N}})$ (see (8.27)), and all *BR*-bending points of $\Delta B_c^{\mathrm{B}}(\varphi)$ in $[\varphi_{\mathrm{R,lwb}}(T_c^{\mathrm{N}}), \varphi_{\mathrm{R,upb}}(T_c^{\mathrm{N}})]$. Determine $\Delta B_c(T_c^{\mathrm{N}})$ using (8.28), i.e.

$$\begin{aligned}
\Delta B_c(T_c^{\mathrm{N}}) &= \min\{\Delta B_p, \min_{\varphi_{\mathrm{R}} \in \Phi^{\mathrm{B}}} \Delta B_c(\varphi_{\mathrm{R}}, T_c^{\mathrm{N}})\} \\
&= \{(8.25)\}\ \min\{\Delta B_p, \min_{\varphi_{\mathrm{R}} \in \Phi^{\mathrm{B}}} (\Delta B_c^{\mathrm{B}}(\varphi_{\mathrm{R}}) + \Delta B_c^{\mathrm{W}}(\varphi_{\mathrm{R}} + T_c^{\mathrm{N}} - T_p))\}.
\end{aligned}$$

The value for $\Delta B_c^{\mathrm{B}}(\varphi_{\mathrm{R}})$ can either be found directly or determined easily by means of interpolation from the *BR*-bending points. The value for $\Delta B_c^{\mathrm{W}}(\varphi_{\mathrm{R}} + T_c^{\mathrm{N}} - T_p)$ can be determined by means of interpolation from the *WO*-bending points. We now find $\Delta B_c(T_c)$ by means of $\Delta B_c(T_c) = \Delta B_c(T_c^{\mathrm{N}}) + k\Delta B_p$.

8.6 Discussion

In this section, we will briefly discuss our results and the application of the techniques presented in this chapter, and reconsider some of the assumptions made in Section 7.1.

8.6.1 Results

In this chapter, we presented techniques to determine both optimistic and pessimistic bounds as well as exact values for the worst-case available amount ΔB_c that can be conditionally guaranteed for strong CGBs on a periodic basis under arbitrary phasing for in-the-place-of resource provision for CGBs and gain time. We illustrated the techniques by means of a single example that we used throughout the chapter.

We will now consider the efficiency of the mechanism by comparing the exact worst-case $\Delta B_c(T_c)$ with its theoretic least upper bound $\Delta B_c^{\text{avg}}(T_c)$, i.e. the amount that becomes available on average as described by (7.1) on page 118. To this end, we distinguish two main cases, based on the periods of ρ_c and ρ_p and the relative phasing φ_{R}.

For the first case, we assume harmonic periods, i.e. $T_c = mT_p$ for $m \in \mathbb{N}^+$, and a relative phasing $\varphi_{\text{R}} = 0$. For this case, $\Delta B_c(T_c) = \Delta B_c^{\text{avg}}(T_c) = m\Delta B_p$, because the relative phasing of every activation of AGB_c is equal to 0, and therefore exactly m budget margins become available as conditionally guaranteed budget during every period T_c in normal mode.

For the second case, we assume arbitrary periods and arbitrary phasings. Based on Lemma 8.5 on page 140, which states the periodicity of $\Delta B_c(T_c)$, we show that the average (periodic) growth of $\Delta B_c(T_c)$ equals the gradient of $\Delta B_c^{\text{avg}}(T_c)$, i.e. $\Delta B_p/T_p$. Hence, the (absolute) difference $\Delta B_c^{\text{avg}}(T_c) - \Delta B_c(T_c)$ is periodic and the ratio $\Delta B_c^{\text{avg}}(T_c)/\Delta B_c(T_c)$ goes to 1 for T_c to infinity. As a consequence, the relative difference $(\Delta B_c^{\text{avg}}(T_c) - \Delta B_c(T_c))/\Delta B_c(T_c)$ is high for small values of T_c and goes to 0 for T_c to infinity. We will now first show that $\Delta B_c(T_c)$ has an average (periodic) growth of $\Delta B_p/T_p$, and than illustrate the absolute difference for our example.

Let $T_c = T'_c + kT_p$, with $T'_c \in [T_c^{\text{EU}}, T_c^{\text{EU}} + T_p)$ and $k \in \mathbb{N}$, hence $T_c \geq T_c^{\text{EU}}$. Based on Lemma 8.5 we derive

$$\begin{array}{rcl}\Delta B_c(T_c) & = & \{\text{Lemma 8.5}\}\ \ \Delta B_c(T'_c) + k\Delta B_p \\ & = & \Delta B_c(T'_c) + \dfrac{T_c - T'_c}{T_p}\Delta B_p \\ & = & \Delta B_c(T'_c) - T'_c\dfrac{\Delta B_p}{T_p} + T_c\dfrac{\Delta B_p}{T_p}.\end{array}$$

Hence, $\Delta B_c(T_c)$ has an average (periodic) growth of $\Delta B_p/T_p$.

For our example, $\Delta B_c^{\text{avg}}(T_c)$ increases linearly from 0 to $T_c^{\text{EU}}\Delta B_p/T_p \approx T_c^{EU}/4 = 29/4 = 6$ for $T_c \in [0, T_c^{\text{EU}}]$, whereas $\Delta B_c(T_c) = 0$ by definition for $T_c \in [0, T_c^{EU}]$. For $T_c = T_c^{\text{EL}}$, we get $\Delta B_c^{\text{avg}}(T_c^{\text{EL}}) = 7\frac{35}{29} \approx 8$, whereas $\Delta B_c(T_c^{\text{EL}}) = 7$ for our example. Hence, the quality improvement that can be achieved for the CGB consumer ρ_c is much smaller than what could be achieved using AGBs for small values of T_c. This is an immediate consequence of the restrictions imposed, i.e.

- we assumed an instantaneous budget configuration change implemented by means of in-the-place-of resource provision, which gives rise to scheduling imperfections;
- we assumed both arbitrary periods and arbitrary phasings;
- we required ΔB_c to become available on a periodic basis at the period and phasing of the CGB consumer.

As mentioned before $\Delta B_c(T_c) = \Delta B_c^{\text{avg}}(T_c)$ for harmonic periods and a relative phasing $\varphi_{\text{R}} = 0$. The absolute difference between ΔB_c and ΔB_c^{avg} may become smaller for arbitrary periods and arbitrary phasings with deferred budget configuration changes or a CGB consumer that can also exploit budgets with other periods and phasings.

8.6.2 Spare capacity

The gap between $\Delta B_c(T_c)$ and $\Delta B_c^{\text{avg}}(T_c)$ gives rise to contributions to the spare capacity in normal mode, which can be provided to RCEs by the spare-capacity manager. Conversely, the existing spare capacity in anticipated mode can be used to increase the conditionally guaranteed budget. This will be illustrated below.

For our example, consider a CGB consumer ρ_c, a period $T_c = 42$, and three quality levels $q_{c,1}, q_{c,2}$, and $q_{c,3}$ with resource requirements 2, 5, and 10, respectively. The maximal amount of absolutely guaranteed budget that can be guaranteed to ρ_c on a periodic basis under arbitrary phasings in anticipated mode is equal to 3. Given the resource requirements of ρ_c, only $q_{c,1}$ can be accommodated under arbitrary phasings in anticipated mode, and therefore B_c becomes 2. Hence, the existing spare capacity in anticipated mode allows for an increase of ρ_c's budget of 1. The amount of ΔB_c that can be conditionally guaranteed to ρ_c based on the budget margin B_p is 7; see Figure 8.13. This amount $B_c + \Delta B_c$ of 9 can accommodate the resource requirements of 5 of quality level $q_{c,2}$, but is insufficient to to accommodate $q_{c,3}$. However, the existing spare capacity in anticipated mode can be used to increase the conditionally guaranteed budget with 1 to 8, giving a total budget of 10 for ρ_c in normal mode, which accommodates the resource requirements of $q_{c,3}$. In summary, the existing spare capacity in anticipated mode is another source for the CGB ΔB_c next to the budget margin ΔB_p. Taking this additional source into account requires a refinement of the analysis.

8.6.3 Using bounds rather than exact values

Just like it is not always required to perform an exact test to determine schedulability, it is also not always required to determine the exact value for ΔB_c. Below, we will give an example illustrating that it is sufficient to determine only the bounds in particular situations.

Let the resource requirements of a first quality level of a CGB consumer ρ_c be such that it can be accommodated given the pessimistic (i.e. lower) $\Delta B_{c,\text{lwb}'}(T_c)$ bound for ΔB_c given in (8.20). Moreover, let the resource requirements of a next (higher) quality level of ρ_c be such that it cannot be accommodated given the optimistic (i.e. upper) bound $\Delta B_{c,\text{upb}'}(T_c)$ for ΔB_c given in (8.21). In this case, the CGB_c can accommodate at most the first quality level, irrespective of the exact value of ΔB_c, and we may therefore restrict ourselves to that pessimistic bound in an admission test.

8.6.4 Best-case and worst-case budgets

In this chapter, we assumed 'in-the-place-of' gain-time provision. As a result, a budget is always depleted by either the RCE 'owning' the budget or another RCE to which the gain time of the owning RCE is provided. Hence, we were able to assume a fixed budget for our analysis. Without in-the-place-of gain-time provision, we can no longer assume that a budget is depleted entirely. Hence, rather than assuming a fixed budget, we have to distinguish best-case budgets and worst-case budgets. In such a situation, best-case advancements are based on best-case budgets, and worst-case advancements on worst-case budgets, which are equal to the former fixed budgets. Because the best-case advancement $BA'_p(t)$ based on best-case budgets is at least equal to the best-case advancement $BA_p(t)$ based on the fixed budgets, the worst-case available $\Delta B_c(T_c)$ will at most remain the same, and typically become less.

8.6.5 Time-offsets

In this chapter, we assumed that the CGB provider ρ_p receives an AGB_p with a budget $B_p + \Delta B_p$. As described in Section 7.6.1, it is also possible to provide ΔB_p as a separate budget, with a relative time-offset of at least $WR_p(B_p)$ with respect to the normal budget. The main advantage of such an approach is the reduction of the jitter with which the budget margin becomes available, thereby increasing the amount of budget that can be conditionally guaranteed. As discussed before, such an approach and its analysis falls outside the scope of this thesis.

8.6.6 EDF

Although we assumed FPPS in this chapter, the approach presented to analyze CGBs is independent of FPPS. The approach is based on response times and occupied times, and therefore equally well applies to a system using EDF. Although a technique to determine worst-case response times for EDF has been given by Spuri [1996], techniques for the other notions do not exist yet for EDF.

9

Supplementary Analytical Case Studies

In this chapter, we present two supplementary case studies illustrating the applicability of the extensions to RMA presented in Chapters 4 and 5.

In Section 9.1, we consider one of the applications of best-case response times, being the analysis of the effect of jitter in a distributed multiprocessor system with task dependencies. In Section 9.2, we present equations and associated procedures to determine exact worst-case response times of periodic tasks under fixed-priority scheduling with deferred preemption (FPDS) and arbitrary phasing. The worst-case response times for FPDS are described in terms of the worst-case response times and worst-case occupied times for FPPS presented in Chapter 4.

9.1 Jitter

In this section, we consider one of the applications of best-case response times, being the analysis of distributed multiprocessor systems. To this end, we first discuss completion jitter and release jitter.

9.1.1 Completion jitter

A task τ_i is released strictly periodically, i.e. release number $k \in \mathbb{Z}$ takes place at time $a_{ik} = \varphi_i + kT_i$. If all executions of τ_i have the same response time, then also the completion times are strictly periodic, i.e. the completion time of execution k

can be written as $f_{ik} = f_i + kT_i$. In general, however, response times vary and, as a consequence, there is variation in the completion times compared to a strictly periodic pattern. The *completion* (or *finalization*) *jitter* $FJ_i(\varphi)$ for a given task τ_i and a particular phasing φ is now defined as the minimum difference $y - x$, for variables $x, y \in \mathbb{R}$ for which the completion times can be captured by

$$kT_i + x \leq f_{ik} \leq kT_i + y, \qquad \text{for all } k \in \mathbb{Z}.$$

In other words, the completion times can be captured in an interval of length $FJ_i(\varphi)$ that is repeated with a period T_i. See Figure 9.1 for an example. The *worst-case*

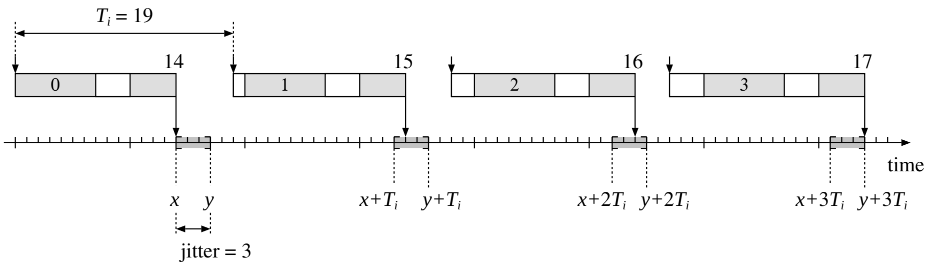


Figure 9.1. An example of completion jitter, due to variations in response times, for task τ_2 of Figure 3.3. The white areas indicate preemptions by task τ_1. Here, the completion jitter equals $y - x = 3$, i.e. the completion times of the executions of the task are captured in an interval of length 3 that is repeated with a period of $T_i = 19$. Note that for this phasing, both $BR_2 = 14$ and $WR_2 = 17$ are taken on.

completion jitter FJ_i of a task τ_i is defined as the maximum completion jitter of any phasing, i.e.

$$FJ_i = \sup_{\varphi} FJ_i(\varphi).$$

If τ_i is released strictly periodically, then an upper bound on its worst-case completion jitter is given by

$$FJ_i \leq WR_i - BR_i, \tag{9.1}$$

as we know that for all φ

$$\varphi_i + kT_i + BR_i \leq f_{ik} \leq \varphi_i + kT_i + WR_i, \qquad \text{for all } k \in \mathbb{Z},$$

and thus $FJ_i(\varphi) \leq WR_i - BR_i$ for all φ. Concerning completion jitter, we note that situations exist in which an execution of a task τ_i having best-case response time BR_i may directly be followed by an execution with a worst-case response time WR_i, and vice versa. An example of this is given in Figure 9.2, where we have two tasks, τ_1 and τ_2, with periods $T_1 = 8$, $T_2 = 12$, and computation times $WC_1 = BC_1 = WC_2 = BC_2 = 4$. For this example, the best-case and worst-case response times of τ_2 are $BR_2 = 4$ and $WR_2 = 8$, respectively, and the figure shows that they indeed can occur right after each other. In this case, the completion jitter of τ_2 equals $FJ_2(\varphi) = FJ_2 = 8-4 = 4$. In general, however, the bound in (9.1) on FJ_i need not be tight, as BR_i and WR_i are not necessarily taken on for the same phasing.

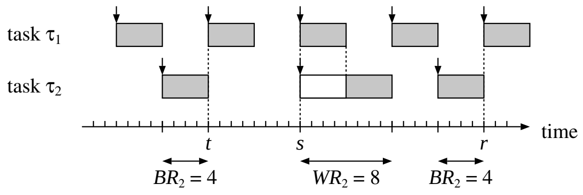


Figure 9.2. A situation in which a best-case execution of τ_2 is directly followed by a worst-case execution, which in turn is directly followed by a best-case execution. Note that times t and r indicate optimal instants for τ_2, and time s indicates a critical instant.

9.1.2 Release jitter

Next to completion jitter, there can also be *activation* (or *release*) *jitter*. In this case, the releases of a task τ_i do not take place strictly periodically, with period T_i, but we assume they take place somewhere in an interval of length AJ_i that is repeated with period T_i. More specifically, the release times a_{ik} satisfy

$$kT_i + x \leq a_{ik} \leq kT_i + y, \qquad \text{for all } k \in \mathbb{Z},$$

for certain $x, y \in \mathbb{R}$ with $y - x = AJ_i$. We now assume $D_i \leq T_i - AJ_i$, which bounds the release jitter from above by $AJ_i \leq T_i - D_i$, since otherwise there may be too little time between two successive releases to complete the task.

In the case of release jitter, the analysis to derive worst-case and best-case response times, as well as completion jitter, is slightly altered. For worst-case response times, it changes as described by Audsley, Burns, Richardson, Tindell & Wellings [1993], Joseph [1996], and Tindell, Burns & Wellings [1994]. The critical instant for task τ_i, i.e. the situation in which an execution of task τ_i is preempted most, is now as depicted in Figure 9.3. If we let execution 0 of task τ_j be the execution of which the release coincides with the release of a worst-case execution of task τ_i at time a, then this worst-case execution is indeed preempted most if release 0 of τ_j takes place rightmost in its interval, and releases 1 and further take place leftmost in their intervals. More precisely, release 0 takes place at time $a_{j0} = y = a$, and release $k = 1, 2, \ldots$ takes place at time $a_{jk} = x + kT_j$. As $x = y - AJ_j$ and $y = a$, this gives $a_{jk} = a - AJ_j + kT_j$ for $k = 1, 2, \ldots$. The number of preemptions of the worst-case execution of τ_i by τ_j is now given by the execution number k of the first release of τ_j at or after the completion of τ_i, which is given by

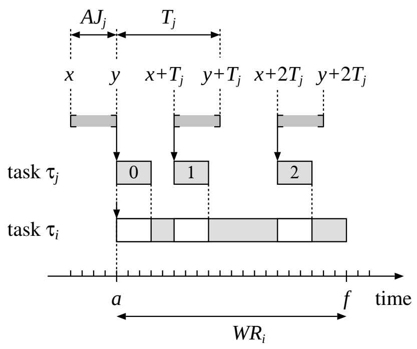


Figure 9.3. A critical instant in the case of release jitter.

the smallest k for which $a_{jk} \geq f$, i.e. for which

$$a - AJ_j + kT_j \geq a + WR_i,$$

or, equivalently,

$$k \geq \frac{WR_i + AJ_j}{T_j}.$$

The smallest value satisfying this inequality is given by

$$k = \left\lceil \frac{WR_i + AJ_j}{T_j} \right\rceil.$$

The remainder of the worst-case analysis stays the same. As a result, the worst-case response times in case of release jitter are computed as follows.

$$\begin{aligned} WR_i^{(0)} &= WC_i \\ WR_i^{(l+1)} &= WC_i + \sum_{j<i} \left\lceil \frac{WR_i^{(l)} + AJ_j}{T_j} \right\rceil WC_j, \quad l = 0, 1, \ldots \end{aligned}$$

For best-case response times, we can accommodate for the effect of release jitter on the analysis in a quite similar way. Consider a best-case execution of task τ_i, which is released at time a and completed at time f. We first note that Lemmas 5.1 and 5.2 also hold in case of release jitter, as in the corresponding proofs we did not use the fact that releases were strictly periodic. Hence, we can use the result of Lemma 5.2, which states that we can determine the amount of preemption of a best-case execution of τ_i by a higher priority task τ_j, by determining the number of releases of τ_j strictly between times a and f. Releases outside this open interval do not cause any preemption, and releases within this interval each cause a preemption by an amount BC_j. So, we only have to revisit the counting of the number of releases of τ_j (strictly) between times a and f.

A situation in which τ_j has a minimal number of releases (strictly) between times a and f is given in Figure 9.4. If we again let release 0 of τ_j be the first

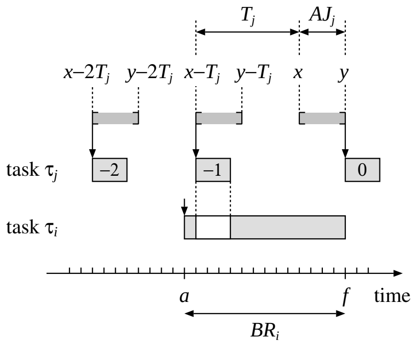


Figure 9.4. An optimal instant in the case of release jitter.

release of τ_j at or after time f, as in the proof of Theorem 5.1, then we have a minimal number of releases of τ_j in the interval (a, f) if release 0 of τ_j takes place rightmost in its release interval, and at time f, and releases $-1,-2,\dots$ take place leftmost in their release intervals. More precisely, the number of releases of τ_j in the interval (a, f) is minimal if $a_{j0} = y = f$ and $a_{jk} = x + kT_j$ for $k = -1,-2,\dots$. As $x = y - AJ_j$ and $y = f$, this gives $a_{jk} = f - AJ_j + kT_j$ for $k = -1,-2,\dots$. The number of preemptions of the best-case execution of τ_i by τ_j, i.e. the number of releases of τ_j in (a, f) is now given by the largest $m \geq 0$ for which $a_{j,-m} > a$, i.e. for which

$$f - AJ_j + (-m)T_j > f - BR_i,$$

or, equivalently,

$$m < \frac{BR_i - AJ_j}{T_j}.$$

The largest non-negative value satisfying this inequality is given by

$$m = \left(\left\lceil \frac{BR_i - AJ_j}{T_j} \right\rceil - 1 \right)^{+}.$$

Here, the notation w^{+} stands for $\max\{w, 0\}$, which is used to indicate that the number of preemptions cannot be negative. As a result, the best-case response times in case of release jitter are computed as follows.

$$\begin{aligned} BR_i^{(0)} &= WR_i \\ BR_i^{(l+1)} &= BC_i + \sum_{j<i} \left(\left\lceil \frac{BR_i^{(l)} - AJ_j}{T_j} \right\rceil - 1 \right)^{+} BC_j, \quad l = 0,1,\dots \end{aligned}$$

Similar results for best-case response times in case of release jitter are given by

Redell & Sanfridson [2002] and Redell [2003].

Given the worst-case and best-case response times of a task τ_i including the effect of release jitter of higher priority tasks, and given the release jitter of τ_i itself, its worst-case completion jitter is now bounded by

$$FJ_i \leq AJ_i + WR_i - BR_i. \tag{9.2}$$

Furthermore, this bound is tight, as the example in Figure 9.5 shows.

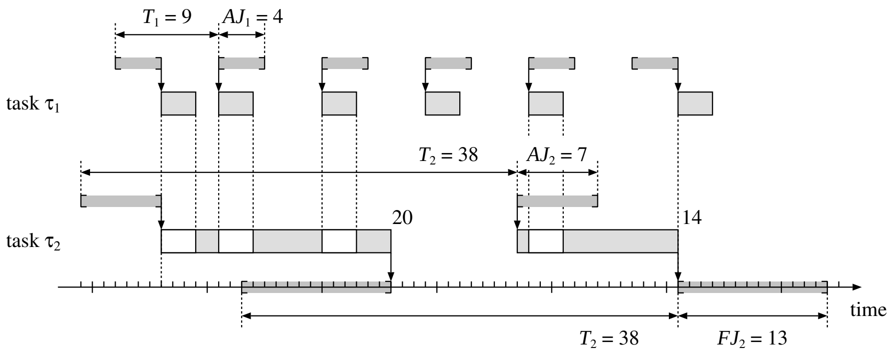


Figure 9.5. An example where equality in the completion jitter bound of (9.2) is achieved for task τ_2. Here, $WR_2 = 20$ and $BR_2 = 14$, and $FJ_2 = FJ_2(\varphi) = 13 = 7 + 20 - 14$.

9.1.3 Distributed multiprocessor systems

In order to take the effect of jitter into account in a distributed multiprocessor system with task dependencies [Palencia Gutiérrez et al., 1998; Kim et al., 2000], where the completion of a task on one processor may trigger the release of a following task on another processor, we can use the following iterative procedure [Palencia Gutiérrez et al., 1998]. We start with an estimated release jitter $AJ_j = 0$ for all tasks τ_j, and do the above calculation of worst-case and best-case response times on each processor. We then determine the completion jitter bound of each task, as given by (9.2). Next, we update the estimate of the release jitter of each task that is triggered by another task, by making it equal to the completion jitter bound of the triggering task. With these new estimates, we then again determine worst-case and best-case response times. We repeat this process until we obtain stable values or until the computed response times exceed their deadlines.

During the above process, the jitter bounds and the worst-case response times increase, and the best-case response times decrease, again causing the jitter bounds to increase, etc. This monotonicity, together with the fact that the response times are bounded, and that they can only take on a finite number of values, implies termination in a finite number of steps. Furthermore, this shows that if we redetermine the worst-case and best-case response times for new estimates of the release jitter, we can use their final values of the previous iteration for initialization.

In general, the newly derived best-case response times lead to tighter bounds on completion jitter and thus also on release jitter of following tasks, in turn resulting in tighter worst-case response time bounds of other tasks. A final remark that we would like to make is that we assumed arbitrary phasings in our analysis. If there are dependencies between tasks on the same processor causing the phasings to have a special structure, then the derived bounds on jitter and response times might still be pessimistic.

9.2 Fixed priority scheduling with deferred preemption

In this section, we present equations and associated procedures to determine the exact worst-case response times of periodic tasks under fixed-priority scheduling with deferred preemption (FPDS) and arbitrary phasing. Our analysis is based on a dedicated conjecture for an ε-critical instant of a task under FPDS and arbitrary phasing. We start this section with a rationale for FPDS. Next we refine our model of Section 3.1 for FPDS. We subsequently address worst-case response times. To this end, we first describe our conjecture for an ε-critical instant. We subsequently present equations for worst-case response times, and illustrate the equations by means of an example. This section is concluded with a discussion, in which we compare our results with related work.

9.2.1 Rationale

Based on the seminal paper of Liu and Layland [1973], many results have been achieved in the area of worst-case analysis for FPPS. Arbitrary preemption of real-time tasks has a number of drawbacks, though. In particular in systems using cache memory, e.g. to bridge the speed gap between processors and main memory, arbitrary preemptions induce additional cache flushes and reloads. As a consequence, system performance and predictability are degraded, complicating system design, analysis and testing [Burns and Wellings, 1997; Gopalakrishnan and Parulkar, 1996; Lee et al., 1998; Simonson and Patel, 1995]. Although fixed-priority non-preemptive scheduling (FPNS) may resolve these problems, it generally leads to reduced schedulability compared to FPPS. Therefore, alternative scheduling schemes have been proposed between the extremes of arbitrary preemption and no preemption. These schemes are also known as deferred preemption or co-operative scheduling [Burns, 1994].

9.2.2 Model

For FPDS, we need to refine our basic model of Section 3.1. Each job of task τ_i is now assumed to consist of $m(i)$ subjobs. Subjob l of τ_i is characterized by a best-case computation time $BC_{i,l} \in \mathbb{R}^+$, where $BC_i = \sum_{l=1}^{m(i)} BC_{i,l}$, and a worst-case computation time $WC_{i,l} \in \mathbb{R}^+$, where $WC_i = \sum_{l=1}^{m(i)} WC_{i,l}$. We assume that subjobs are non-preemptable. Hence, tasks can only be preempted at subjob boundaries, i.e. at so-called *preemption points*. For convenience, we will use the term WF_i to denote the worst-case computation time $WC_{i,m(i)}$ of the final subjob of task τ_i. Note that when $m(i) = 1$ for all i, we have FPNS as special case.

In order to be able to relate the response times for different scheduling approaches, we use the superscript P to denote FPPS (similarly, we will use superscripts D and N later to denote FPDS and FPNS, respectively), and parameterize the response times and occupied times of task τ_i with its computation time.

9.2.3 A critical instant

The non-preemptive nature of subjobs may cause blocking of a task by at most one lower priority task under FPDS. The worst-case blocking time WB_i of task τ_i by a lower priority task is equal to the longest worst-case computation time of any subjob of a task with a priority lower than task τ_i, which is given by

$$WB_i = \max_{j>i} \max_{1 \le l \le m(j)} WC_{j,l}. \tag{9.3}$$

In order to determine worst-case response times under arbitrary phasing, we have to revisit critical instants. In this thesis, we merely postulate the following conjecture.

Conjecture 9.1. *An ε-critical instant of a task τ_i under FPDS and arbitrary phasing occurs when that task is released simultaneously with all tasks with a higher priority than τ_i, and the subjob with the longest computation time of all lower priority tasks starts an infinitesimal time $\varepsilon > 0$ before that simultaneous release.* $\square$

From this conjecture we conclude that a critical instant for FPDS is a supremum for all but the lowest priority task, i.e. that instant cannot be assumed.

9.2.4 Equations for worst-case response times

For the analysis, we consider three cases: the highest priority task τ_1, the lowest priority task τ_n, and a medium priority task τ_i (with $1 < i < n$). The highest priority task τ_1 may be blocked, but is never preempted. The worst-case response time WR_1^{D} of task τ_1 therefore includes a term WB_1, i.e.

$$WR_1^{\mathrm{D}} = WB_1 + WC_1. \tag{9.4}$$

Note that $WB_1 + WC_1$ is a supremum, i.e. that value cannot be assumed. Further note that this latter equation may also be written as $WR_1^{\mathrm{D}} = WR_1^{\mathrm{P}}(WB_1 + WC_1)$, or $WR_1^{\mathrm{D}} = WR_1^{\mathrm{P}}(WB_1 + WC_1 - WF_1) + WF_1$. Because $WR_1^{\mathrm{P}} = WO_1^{\mathrm{P}}$, the equation may even be written as $WR_1^{\mathrm{D}} = WO_1^{\mathrm{P}}(WB_1 + WC_1)$, or $WR_1^{\mathrm{D}} = WO_1^{\mathrm{P}}(WB_1 + WC_1 - WF_1) + WF_1$.

The lowest priority task τ_n may be preempted (at subjob boundaries), but is never blocked. The worst-case response time WR_n^{D} of task τ_n can be found by calculating the worst-case start time of the final subjob, and adding its computation time F_n. The non-preemptive nature of the other subjobs of τ_n may result in deferred preemptions by higher priority tasks. Although that has an influence on the order of the executions of the subjobs of tasks, it does not influence the total amount of time spent on those executions. The amount of time spent on executions of all but the final subjob of τ_n including the (deferred) preemptions of higher priority tasks is given by $WR_n^{\mathrm{P}}(WC_n - WF_n)$. The final subjob of τ_n may subsequently start after the aligning successive executions of higher priority tasks have completed. Hence, the worst-case start time of the final subjob of task τ_n is given by $WO_n^{\mathrm{P}}(WC_n - WF_n)$, and we arrive at

$$WR_n^{\mathrm{D}} = WO_n^{\mathrm{P}}(WC_n - WF_n) + WF_n. \tag{9.5}$$

Note that $WO_n^{\mathrm{P}}(WC_n - WF_n) + WF_n$ is a maximum, i.e. that value can be assumed. Further note that for $m(n) = 1$, we get $WO_n^{\mathrm{P}}(WC_n - WF_n) = WO_n^{\mathrm{P}}(0)$, which is equal to the worst-case start time of task τ_n.

A medium priority task τ_i with $1 < i < n$, may be both preempted at subjob boundaries by higher priority tasks and blocked by one lower priority task. Similar to the lowest priority task, the worst-case response time of τ_i can be found by calculating the worst-case start time of the final subjob, and by subsequently adding its computation time WF_i. Similarly to the lowest priority task, the non-preemptive nature of the other subjobs of τ_i has no influence on the worst-case start time of the final subjob. At first hand, it therefore looks as if the same reasoning applies as for the lowest priority task, and that we can calculate the worst-case start time by means of $WO_i^{\mathrm{P}}(WB_i + WC_i - WF_i)$. However, the blocking subjob of the lower priority tasks has to start an infinitesimal time $\varepsilon > 0$ *before* the simultaneous release of τ_i and its higher priority tasks. Hence, the amount of time spent from the release of τ_i on executions of the blocking subjob and all but the final subjob of τ_i including the (deferred) preemptions of higher priority tasks is given by $WO_i^{\mathrm{P}}(B_i - \varepsilon + WC_i - WF_i)$. This latter value is equal to $WR_i^{\mathrm{P}}(WB_i + WC_i - WF_i)$ *minus* an infinitesimal time $\varepsilon > 0$. It is exactly this infinitesimal difference, which approaches zero, that allows the final subjob of τ_i to start executing, and defers potential additional preemptions from higher priority tasks at time $WR_i^{\mathrm{P}}(WB_i + WC_i - WF_i)$. The worst-case response time WR_i^{D} of a medium priority task τ_i is therefore given by

$$WR_i^{\mathrm{D}} = WR_i^{\mathrm{P}}(WB_i + WC_i - WF_i) + WF_i. \tag{9.6}$$

Note that $WR_i^{\mathrm{P}}(WB_i + WC_i - WF_i) + WF_i$ is also a supremum.

The impact of blocking and preemption on the worst-case response time of a

Table 9.1. Impact on worst-case response time of a task τ_i under fixed-priority scheduling with deferred preemption.

	$i=1$	$1<i<n$	$i=n$
blocking	yes	yes	no
preemption	no	yes	yes

task τ_i for $1 \le i \le n$ under FPDS and arbitrary phasing is summarized in Table 9.1. Because this impact differs for $i = 1$, $1 < i < n$, and $i = n$, the equations for WR_i^{D} also differ. An overview of the equations is given below.

$$WR_i^{\mathrm{D}} = \begin{cases} WB_1 + WC_1 & \text{for } i = 1 \\ WR_i^{\mathrm{P}}(WB_i + WC_i - WF_i) + WF_i & \text{for } 1 < i < n \\ WO_n^{\mathrm{P}}(WC_n - WF_n) + WF_n & \text{for } i = n \end{cases} \tag{9.7}$$

As mentioned above, $WR_1^{\mathrm{P}} = WO_1^{\mathrm{P}}$, and we may therefore rewrite the equation for $i=1$ to $WR_1^{\mathrm{D}} = WR_1^{\mathrm{P}}(WB_1 + WC_1 - WF_1) + WF_1$ as well as to $WR_1^{\mathrm{D}} = WO_1^{\mathrm{P}}(WB_1 + WC_1 - WF_1) + WF_1$. Hence, the equation for $i = 1$ is similar to the equation for both $1 < i < n$ and $i = n$.

9.2.5 An example

To illustrate the equations, consider the task characteristics of Table 9.2. For FPNS, the set is not schedulable because the worst-case response time WR_1^{N} of task τ_1 is equal to $WB_1 + WC_1 = 4 + 2 = 6$, which exceeds the task's deadline. The worst-case response time WR_1^{N} of task τ_1 would become equal to the deadline D_1 for $WB_1 = 2$. For FPDS, let $m(1) = 1$, $m(2) = 2$ with $WC_{2,1} = 1$ and $WC_{2,2} = 2$, and $m(3) = 2$ with $WC_{3,1} = WC_{3,2} = 2$; see Table 9.2. Using the equations above

Table 9.2. Task characteristics and worst-case response times under FPPS, FPDS, and FPNS.

	T_i	D_i	WC_i	WR_i^{P}	WR_i^{D}	WR_i^{N}
τ_1	5	4	2	2	4	6
τ_2	7	7	1 + 2	5	7	13
τ_3	30	30	2 + 2	28	21	16

yields $WR_1^{\mathrm{D}} = WB_1 + WC_1 = 2+2 = 4$, $WR_2^{\mathrm{D}} = WR_2^{\mathrm{P}}(WB_2 + WC_2 - WF_2) + WF_2 = WR_2^{\mathrm{P}}(2+3-2)+2 = 7$, and $WR_3^{\mathrm{D}} = WO_3^{\mathrm{P}}(C_3 - WF_3) + WF_3 = WO_3^{\mathrm{P}}(2) + 2 = 21$. Hence, by splitting both task τ_2 and task τ_3 into two non-preemptive subjobs, the task set becomes schedulable under FPDS.

Note that FPDS has two opposite influences on the worst-case response time WR_i of a task τ_i when compared with FPPS. When τ_i itself is (partially) non-preemptive, WR_i (possibly) decreases (in the example $WR_3^{\mathrm{D}} < WR_3^{\mathrm{P}}$), because the execution of the final subjob of the task is not preempted by higher priority tasks. When a task τ_k with a lower priority than task τ_i is (partially) non-preemptive, WR_i increases due to the blocking nature of the non-preemptiveness of task τ_k, causing $WR_1^{\mathrm{D}} > WR_1^{\mathrm{P}}$ in the example. Similar observations hold for FPNS compared to FPDS.

9.2.6 Discussion

We briefly compare our results for worst-case response times on FPDS with those presented in the literature.

The schedulability test described by Gopalakrishnan & Parulkar [1996] is based on utilization bounds, and is therefore typically pessimistic. The worst-case response time analysis presented by Lee et al. [1998] is based on a single equation, i.e. it is uniform for all tasks. The blocking effect of (partially) non-preemptive lower priority tasks has been covered in that analysis, but the effect of the non-preemptive nature of the final subjob is not taken into account. The analysis is therefore pessimistic. Note that the main focus of [Lee et al., 1998] is on incorporating the effect of cache related preemption delays in worst-case response times, an aspect we do not address in our analysis.

The results presented by Burns [1994], Burns [2001], and Burns & Wellings [1997] are very similar to ours. Unlike our approach, their approach is uniform for all tasks. Using our notation, the worst-case response time $\tilde{WR}_i^{\mathrm{D}}$ under FPDS and arbitrary phasing presented by Burns [1994] and Burns & Wellings [1997] is given

$$\tilde{WR}_i^{\mathrm{D}} = WR_i^{\mathrm{P}}(WB_i + WC_i - (WF_i - \Delta)) + (WF_i - \Delta).$$

According to [Burns & Wellings, 1997], Δ is an arbitrary small positive value needed to ensure that the final subjob has actually started. Hence, when task τ_i has consumed $WC_i - (WF_i - \Delta)$, the final subjob has (just) started. When Δ approaches to zero, we may rewrite the above equation to

$$\tilde{WR}_i^{\mathrm{D}} = WO_i^{\mathrm{P}}(WB_i + WC_i - WF_i) + WF_i.$$

This result is identical to ours for the highest and lowest priority tasks, but differs from ours for intermediate tasks. For the example presented above, our analysis yields $WR_2^{\mathrm{D}} = 7$, whereas the analysis presented by Burns & Wellings [1997] yields $\tilde{WR}_2^{\mathrm{D}} = 9$. In the example, this latter result is too pessimistic; because $\tilde{WR}_2^{\mathrm{D}}$ exceeds the deadline of τ_2, the task set would incorrectly be considered non-schedulable. The difference between our analysis and the analysis presented in [Burns & Wellings, 1997] can be traced back to Conjecture 9.1. In that conjecture, we state that the subjob with the longest computation time of all lower priority tasks starts an infinitesimal time $\varepsilon > 0$ *before* the simultaneous release of τ_i and all tasks with a higher priority than τ_i. The analysis presented by Burns & Wellings [1997] does not take into account that a task τ_i can only be blocked by a subjob of a lower priority task if that subjob also starts an amount of time Δ before the simultaneous release of τ_i and all tasks with a higher priority than τ_i. When this aspect is taken into account in the analysis presented by Burns & Wellings [1997], e.g. when WB_i is replaced by $WB_i - \Delta$ in the first equation for $\tilde{WR}_i^{\mathrm{D}}$, their result becomes identical to ours.

In the scheduling analysis review presented by Burns [2001], the worst-case response time $\tilde{WR}_i^{\mathrm{D}}$ ignores the term Δ, i.e.

$$\tilde{WR}_i^{\mathrm{D}} = WR_i^{\mathrm{P}}(WB_i + WC_i - WF_i) + WF_i.$$

This result is identical to ours, except for the lowest priority task. For the example presented above, this results in $\tilde{WR}_3^{\mathrm{D}} = WR_3^{\mathrm{P}}(WC_3 - WF_3) + WF_3 = 16$, which is too optimistic.

10

Conclusion

In this thesis we studied real-time scheduling for cost-effective media processing in software using conditionally guaranteed budgets (CGBs). Unlike a normal budget, which has an absolute guarantee, a CGB can only be allocated with a conditional guarantee based on a conditional admission test. CGBs may be viewed as a refinement of the resource reservation technique as currently supported by existing resource kernels. We presented a design for CGBs, and an accompanying implementation as an extension to an existing resource manager, the so-called budget scheduler. Because the budget scheduler is based on a fixed-priority scheduling (FPS) model, so is the implementation of our CGBs, and we therefore presented a conditional admission test based on rate monotonic analysis (RMA). For this admission test we conceived various extensions to RMA.

We presented the context of this work by giving a circumstantial description of a co-operative QoS approach to media processing in software in high volume electronics multimedia consumer terminals. This QoS approach combines application adaptation, a multi-layer control hierarchy, and a reservation-based resource manager that provides guaranteed CPU budgets. Moreover, the approach aims at close-to-average resource allocation for reasons of cost-effectiveness. This combination gives rise to a problem related to user focus. Upon a structural load increase of an application with user focus, its output quality has a dip. Because stable output is a primary requirement for an application with user focus, this dip is referred to as the user-focus problem. CGBs have been conceived to resolve this problem. CGBs are a mechanism at the level of the resource manager, that facilitate instantaneous budget configuration changes for anticipated changes. Hence, when an application with user focus claims additional, anticipated amounts of resources, those resources can be provided instantaneously, and the output quality of that application can therefore remain stable upon anticipated structural load increases. CGBs can be exploited by policies in the control hierarchy to improve the cost-effectiveness of the reservation-based system using controlled quality improvements. Being a general mechanism, CGBs can be used in other contexts as well. This has been illustrated by examples, such as emergency applications.

This thesis presents two main variants of CGBs: *weak* CGBs and *strong* CGBs. A weak CGB is based on gain time of a CGB provider, and can therefore only be weakly guaranteed, even when that gain time becomes available consistently. Strong CGBs are based on the assumption that a structural load increase can be detected timely, e.g. as a by-product of the normal behavior of application-domain specific control. A strong CGB is based on restrained budget use of a CGB provider, i.e. with appropriate means for detecting structural load increases, a CGB provider can explicitly *claim* and *release* anticipated amounts of additional resources. Those additional resources for a CGB provider are the basis for a CGB that is provided to a CGB consumer, and the CGB consumer is explicitly informed about the availability of its CGB. Because a CGB consumer can count on its CGB between the release and subsequent re-claim of the CGB provider, this variant of CGBs is termed a strong CGB.

We presented the so-called concept of in-the-place-of CGB provision that accommodates instantaneous budget configuration changes, and covers both variants of CGBs. We showed that this concept gives rise to scheduling imperfections, not only for FPS, but also for the earliest deadline first (EDF) scheduling algorithm. Accompanying implementations for CGBs have been described as extensions of the budget scheduler. We also presented the concept of in-the-place-of gain-time provision with an accompanying implementation. The mechanisms for weak CGBs and in-the-place of gain-time provision are inherently similar. Conversely, strong CGBs and in-the-place-of gain-time provision are orthogonal, and although the mechanisms are similar, we showed that they can be applied independently.

As a basis for the analysis of CGBs, we presented various extensions to RMA, in particular *best-case response times* next to worst-case response times, and best-case and worst-case *occupied times*. For each of these notions, we derived a recursive equation and presented an iterative procedure to find its solution. Moreover, we studied the efficiency of these iterative procedures. We showed that the number of iterations needed to determine worst-case response times using a standard initial value increases logarithmically for increasing worst-case computation times. We proved that the number of iterations for an alternative initial value is periodic and bounded by a constant.

For the admission test for CGBs, we determined the worst-case amount of budget that can be conditionally guaranteed for CGBs on a periodic basis under arbitrary phasings for in-the-place-of CGB provision for fixed-priority preemptive scheduling (FPPS). We presented techniques to determine both optimistic and pessimistic bounds as well as exact values for that amount, and an efficient algorithm to determine this exact amount.

Finally, we showed that the extensions to RMA can also be applied in other contexts, such as jitter analysis and exact worst-case response time analysis for fixed-priority scheduling with deferred preemption (FPDS).

Bibliography

AARTS, E., R. HARWIG, AND M. SCHUURMANS [2001], Ambient intelligence, in: P.J. Denning (ed.), *The invisible future: The seamless integration of technology into everyday life*, McGraw-Hill, Inc., New York, NY, USA, 235–250.

AUDSLEY, N.C., A. BURNS, R.I. DAVIS, K.W. TINDELL, AND A.J. WELLINGS [1995], Fixed priority pre-emptive scheduling: An historical perspective, *Journal of Real-time Systems* **8**, 173–198.

AUDSLEY, N.C., A. BURNS, M.F. RICHARDSON, K. TINDELL, AND A.J. WELLINGS [1993], Applying new scheduling theory to static priority pre-emptive scheduling, *Software Engineering Journal* **8**:5, 284–292.

AUDSLEY, N.C., A. BURNS, M.F. RICHARDSON, AND A.J. WELLINGS [1991], Hard real-time scheduling: The deadline monotonic approach, *Proc. 8th IEEE Workshop on Real-Time Operating Systems and Software (RTOSS)*, 133–137.

AUDSLEY, N.C., A. BURNS, M.F. RICHARDSON, AND A.J. WELLINGS [1993], Incorporating unbounded algorithms into predictable real-time systems, *Computer Systems Science & Engineering* **8**:2, 80–89.

AUDSLEY, N.C., R.I. DAVIS, AND A. BURNS [1994], Mechanisms for enhancing the flexibility and utility of hard real-time systems, *Proc. 15th IEEE Real-Time Systems Symposium (RTSS)*, 12–21.

BAKER, T.P. [1991], Stack-based scheduling of realtime processes, *Real-Time Systems* **3**:1, 67–99.

BARUAH, S.K., L.E. ROSIER, AND R.R. HOWELL [1990], Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor, *Real-Time Systems* **2**, 301–324.

BINI, E., AND G.C. BUTTAZZO [2002], The space of rate monotonic schedulability, *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS)*, 169–178.

BINI, E., G. BUTTAZZO, AND G. BUTTAZZO [2001], A hyperbolic bound for the rate monotonic algorithm, *Proc. 13th Euromicro Conference on Real-Time Systems (RTSS)*, 59–66.

BRIL, R.J., M. GABRANI, C. HENTSCHEL, G.C. VAN LOO, AND E.F.M. STEFFENS [2001], QoS for consumer terminals and its support for product families, *Proc. International Conference on Media Futures (ICMF)*, 299–302.

BRIL, R.J., C. HENTSCHEL, E.F.M. STEFFENS, M. GABRANI, G.C. VAN LOO, AND J.H.A. GELISSEN [2001], Multimedia QoS in consumer terminals (invited lecture), *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, 332–343.

BRIL, R.J., E.-J.D. POL, AND W.F.J. VERHAEGH [2002], An initial value for on-line response time calculations, *Proc.* 1st *Philips Symposium on Intelligent Algorithms (SOIA)*, 105–117, http://www.extra.research.philips.com/publ/nl-ms/r-22.467.pdf.

BRIL, R.J., AND E.F.M. STEFFENS [2001], User focus in consumer terminals and conditionally guaranteed budgets, *Proc.* 9th *International Workshop on Quality of Service (IWQoS)*, 107–120, Lecture Notes in Computer Science (LNCS) 2092, Springer-Verlag.

BRIL, R.J., E.F.M. STEFFENS, G.C. VAN LOO, M. GABRANI, AND C. HENTSCHEL [2001], Dynamic behavior of consumer multimedia terminals: System aspects, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 597–600.

BRIL, R.J., E.F.M. STEFFENS, AND W.F.J. VERHAEGH [2001], Best-case response times of real-time tasks, *Proc.* 2nd *Philips Workshop on Scheduling and Resource Management (SCHARM)*, 19–27, http://www.extra.research.philips.com/publ/nl-ms/r-20.914.pdf.

BRIL, R.J., E.F.M. STEFFENS, AND W.F.J. VERHAEGH [2004], Best-case response times and jitter analysis of real-time tasks, *Journal of Scheduling* **7**:2, 133–147.

BRIL, R.J., AND W.F.J. VERHAEGH [2003], *Analysis complementing RMA - Part I: Basic equations and procedures*, Nat.lab. technical note 2001/166, Philips Research, http://www.extra.research.philips.com/publ/rep/nl-ur/ur2001-166.pdf.

BRIL, R.J., W.F.J. VERHAEGH, AND J.J. LUKKIEN [2004], Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption, *Proc. Work-in-Progress (WiP) session of the* 16th *Euromicro Conference on Real-Time Systems (ECRTS), Technical Report from the University of Nebraska-Lincoln, Department of Computer Science and Engineering (TR-UNL-CSE-2004-0010)*, 57–60.

BRIL, R.J., W.F.J. VERHAEGH, AND E.-J.D. POL [2003], Initial values for on-line response time calculations, *Proc.* 15th *Euromicro Conference on Real-Time Systems (ECRTS)*, 13–22.

BRUNHEROTO, J., R. CHEWRNOCK, P. DETTORI, X. DONG, J. PARASZCZAK, F. SCHAFFA, AND D. SEIDMAN [2000], Issues in data embedding and synchronization for digital television, *Proc. IEEE International Conference* *on Multimedia and Expo (ICME), Vol. 3*, 1233–1236.

BURNS, A. [1991], Scheduling hard real-time systems: a review, *Software Engineering Journal* **6**:3, 116–128.

BURNS, A. [1994], Preemptive priority based scheduling: An appropriate engineering approach, in: S. Son (ed.), *Advances in Real-Time Systems*, Prentice-Hall, 225–248.

BURNS, A. [2001], Defining new non-preemptive dispatching and locking policies for Ada, *Proc. 6th Ada-Europe International Conference*, 328–336.

BURNS, A., AND A.J. WELLINGS [1997], Restricted tasking models, *Proc. 8th International Real-Time Ada Workshop*, 27–32.

BUTTAZZO, G.C. [2002], *Hard Real-time Computing Systems - Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, fourth printing.

CACCAMO, M., G. BUTTAZZO, AND L. SHA [2000], Capacity sharing for overrun control, *Proc. 21st Real-Time Systems Symposium (RTSS)*, 295 – 304.

FOSTER, I., A. ROY, AND V. SANDER [2000], A Quality of Service architecture that combines resource reservations and application adaptation, *Proc. 8th International Workshop on Quality of Service (IWQoS)*, 181–188.

GABRANI, M., C. HENTSCHEL, L. STEFFENS, AND R.J. BRIL [2001], Dynamic behavior of consumer multimedia terminals: Video processing aspects, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 1220–1223.

GABRANI, M., C. HENTSCHEL, L. STEFFENS, AND R.J. BRIL [2003], System aspects of deploying scalable video algorithms in multimedia consumer terminals, *Submitted for publication*.

GELISSEN, J.H.A. [2001], The ITEA project EUROPA: A software platform for digital CE appliances, *Proc. 7th International Symposium on Broadcasting Technology (ISBT)*, 17–23.

GEORGE, L., N. RIVIERRE, AND M. SPURI [1996], *Preemptive and non-preemptive real-time uni-processor scheduling*, Technical Report 2966, Institut National de Recherche et Informatique et en Automatique (INRIA), France.

GONZÁLEZ HARBOUR, M., M.H. KLEIN, AND J.P. LEHOCZKY [1994], Timing analysis for fixed-priority scheduling of hard real-time systems, *IEEE Transactions on Software Engineering (TSE)* **20**:1, 13–28.

GOPALAKRISHNAN, R., AND G.M. PARULKAR [1996], Bringing real-time scheduling theory and practice closer for multimedia computing, *Proc. ACM Sigmetrics Conf. on Measurement & modeling of computer systems*, 1–12.

GRAN, C., AND A. SCHELLER [2000], From proven office technologies to the intelligent multimedia home, *Proc. IEEE International Conference on Mul-*

timedia and Expo (ICME), Vol. 3, 1225–1228.

GROBA, A.M., A. ALONSO, J.A. RODRÍQUES, AND M. GARCÍA VALLS [2002], Response time of streaming chains: Analysis and results, *Proc.* 14th *Euromicro Conference on Real Time Systems (ECRTS)*, 182–189.

HAAN, G. DE [1992], *Motion estimation and compensation*, Ph.D. thesis, Technical University of Eindhoven, The Netherlands, ISBN 90-74445-01-2.

HAAN, G. DE [2000], *Video Processing for multimedia systems*, University Press Eindhoven.

HAMANN, C.-J., J. LÖSER, L. REUTHER, S. SCHÖNBERG, J. WOLTER, AND H. HÄRTIG [2001], Quality-assuring scheduling - using stochastic behavior to improve resource utilization, *Proc.* 22nd *IEEE Real-Time Systems Symposium (RTSS)*, 119–129.

HARRIS, J.W., AND H. STOCKER [1998], *Handbook of Mathematics and Computational Science*, Springer-Verlag New York, Inc.

HASKELL, B.G., A. PURI, AND A.N. NETRAVALI [1997], *Digital video: an introduction to MPEG-2*, Chapman & Hall, ITP (International Thomson Publishing), ISBN 0-412-08411-2.

HENTSCHEL, C. [1998], *Video-Signalverarbeitung*, B.G. Teubner Stuttgart.

HENTSCHEL, C., R. BRASPENNING, AND M. GABRANI [2001], Scalable algorithms for media processing, *Proc. International Conference on Image Processing (ICIP)*, 342–345.

HENTSCHEL, C., R.J. BRIL, AND Y. CHEN [2001], How to add video applications to fully loaded consumer terminals, Handout at the International Broadcasting Convent (IBC), (Amsterdam, The Netherlands).

HENTSCHEL, C., R.J. BRIL, AND Y. CHEN [2002], Video Quality-of-Service for multimedia consumer terminals - an open and flexible system approach (invited paper), *Proc. International Conference on Communications, Circuits and Systems (ICCCS), Vol. 1: Communication Theory and Systems*, 659–663.

HENTSCHEL, C., R.J. BRIL, Y. CHEN, R. BRASPENNING, AND T.-H. LAN [2002], Video quality-of-service for consumer terminals - a novel system for programmable components, *Digest of technical papers International Conference on Consumer Electronics (ICCE)*, 28–29.

HENTSCHEL, C., R.J. BRIL, Y. CHEN, R. BRASPENNING, AND T.-H. LAN [2003], Video Quality-of-Service for consumer terminals - a novel system for programmable components, *IEEE Transactions on Consumer Electronics* **49**:4, 1367–1377.

HERMANT, J.-F., L. LEBOUCHER, AND N. RIVIERRE [1996], *Real-time fixed and dynamic priority driven scheduling algorithms: theory and practice*, Technical Report 3081, Institut National de Recherche et Informatique et en

Automatique (INRIA), France.

HUANG, J., P.-J. WAN, AND D.-Z. DU [1998], Criticality- and QoS-based multiresource negotiation and adaptation, *Journal of Real-Time Systems* **15**:3, 249–273.

ISOVIĆ, D., G. FOHLER, AND L. STEFFENS [2003], Timing constraints of MPEG-2 decoding for high quality video: Misconceptions and realistic assumptions, *Proc.* 15th *Euromicro Conference on Real-Time Systems (ECRTS)*, 73–82.

ITU-T [1994], *Recommendation E.800*, Technical report, International Telecommunication Union, http://www.itu.int/home/index.html.

JASPERS, E.G.T. [2003], *Architecture Design of Video Processing Systems on a Chip*, Phd-thesis, Eindhoven University of Technology.

JOSEPH, M. (ed.) [1996], *Real-Time Systems: Specification, Verification and Analysis*, Prentice Hall.

JOSEPH, M., AND P. PANDYA [1986], Finding response times in a real-time system, *The Computer Journal* **29**:5, 390–395.

KIM, T., J. LEE, H. SHIN, AND N. CHANG [2000], Best case response time analysis for improved schedulability analysis of distributed real-time tasks, *Proc. ICDCS Workshop on Distributed Real-Time Systems*, B14–B20.

KLEIN, M.H., T. RALYA, B. POLLAK, R. OBENZA, AND M. GONZÁLEZ HARBOUR [1993], *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers.

LAFRUIT, G., L. NACHTERGALE, K. DENOLF, AND J. BORMANS [2000], 3D computational graceful degradation, *Proc. IEEE International Symposium on Circuits and Systems (ISCAS), Vol. 3*, 547–550.

LAN, T.-H., Y. CHEN, AND Z. ZHONG [2001], MPEG-2 decoding complexity regulation for a media processor, *Proc. IEEE Workshop on Multimedia and Signal Processing (MMSP)*, 193–198.

LEE, C., J. LEHOCZKY, R. RAJKUMAR, AND D. SIEWIOREK [1999a], On quality of service optimization with discrete QoS options, *Proc.* 5th *IEEE Real-time Technology and Applications Symposium (RTAS)*, 276–286.

LEE, C., J. LEHOCZKY, R. RAJKUMAR, AND D. SIEWIOREK [1999b], A scalable solution to the multi-resource QoS problem, *Proc.* 20th *IEEE Real-Time Systems Symposium (RTSS)*, 315–326.

LEE, S., C.-G. LEE, M. LEE, S.L. MIN, AND C.-S. KIM [1998], Limited preemptible scheduling to embrace cache memory in real-time systems, *Proc. ACM Sigplan Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 51–64, Lecture Notes in Computer Science (LNCS) 1474.

LEHOCZKY, J.P. [1990], Fixed priority scheduling of periodic task sets with arbitrary deadlines, *Proc. 11th IEEE Real-Time Systems Symposium (RTSS)*, 201–209.

LEHOCZKY, J.P., L. SHA, AND Y. DING [1989], The rate monotonic scheduling algorithm: Exact characterization and average case behavior, *Proc. 10th IEEE Real-Time Systems Symposium (RTSS)*, 166–171.

LEHOCZKY, J.P., L. SHA, J.K. STROSNIDER, AND H. TOKUDA [1991], Fixed priority scheduling theory for hard real-time systems, in: A.M. van Tilborg and G.M. Koob (eds.), *Foundations of real-time scheduling: scheduling and resource management*, Kluwer Academic Publ., 1–30.

LEUNG, J.Y.T., AND M.L. MERRILL [1980], A note on preemptive scheduling of periodic, real-time tasks, *Information Processing Letters (IPL)* **11**:3, 115–118.

LEUNG, J.Y.T., AND J. WHITEHEAD [1982], On the complexity of fixed-priority scheduling of periodic, real-time tasks, *Performance Evaluation* **2**:4, 237–250.

LI, B., AND K. NAHRSTEDT [1999], Dynamic reconfiguration for complex multimedia applications, *Proc. International Conference on Multimedia Computing and Systems (ICMCS), Vol. 1*, 165–170.

LIPARI, G., AND S. BARUAH [2000], Greedy reclamation of unused bandwidth in constant-bandwidth servers, *Proc. 12th Euromicro Conference on Real-Time Systems (ECRTS)*, 193–200.

LIPPENS, P., B. DE LOORE, G. DE HAAN, P. EECKHOUT, H. HUIJGEN, A. LÖNING, B. MCSWEENEY, M. VERSTRAELEN, B. PHAM, AND J. KETTENIS [1996], A video signal processor for motion-compensated field-rate upconversion in consumer television, *IEEE Journal of Solid-State Circuits* **31**:11, 1762–1769.

LIU, C.L., AND J.W. LAYLAND [1973], Scheduling algorithms for multiprogramming in a real-time environment, *Journal of the ACM* **20**:1, 46–61.

LIU, J.W.S. [2000], *Real-Time Systems*, Prentice Hall.

LONCZEWSKI, F., AND R. JAEGER [2000], An extensible set-top-box architecture for interactive and broadcast services offering sophisticated user guidance, *Proc. IEEE International Conference on Multimedia and Expo (ICME), Vol. 3*, 1403–1406.

MÄKI-TURJA, J., AND M. NOLIN [2004], Tighter response-times for tasks with offsets, *Accepted for the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA).*

MERCER, C.W., S. SAVAGE, AND H. TOKUDA [1994], Processor capability reserves: Operating system support for multimedia applications, *Proc. International Conference on Multimedia Computing and Systems (ICMCS)*, 90–99.

MIGUEL, M.A. DE, J.F. RUIZ, AND M. GARCÍA [2002], Qos-aware component frameworks, *Proc.* 10th *IEEE International Workshop on Quality of Service (IWQoS)*, 161–169.

MOK, A.K.-L. [1983], *Fundamental design problems of distributed systems for the hard-real-time environment*, Phd thesis, Massachusetts Institute of Technology, http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-297.pdf.

MORROS, J.R., AND F. MARQUÉS [1999], A proposal for dependent optimization in scalable region-based coding systems, *Proc. IEEE International Conference on Image Processing (ICIP), Vol. 4*, 295–299.

NAHRSTEDT, K., H. CHU, AND S. NARAYAN [1998], QoS-aware resource management for distributed multimedia applications, *Journal on High-Speed Networking, Special Issue on Multimedia Networking* **8**:3-4, 227–255.

NIZ, D. DE, L. ABENI, S. SAEWONG, AND R. RAJKUMAR [2001], Resource sharing in reservation-based systems, *Proc.* 22nd *Real-Time Systems Symposium (RTSS)*, 171–180.

NIZ, D. DE, S. SAEWONG, R. RAJKUMAR, AND L. ABENI [2001], Resource sharing in reservation-based systems, *Proc.* 7th *Real-Time Technology and Applications Symposium (RTAS)*, 130–131.

OBENZA, R. [1994], Guaranteeing real-time performance using RMA, *Embedded Systems Programming*, 26 – 40.

OMMERING, R.C. VAN [1998], Koala, a component model for consumer electronics product software, *Proc.* 2nd *Int. ESPRIT ARES Workshop*, 76–86, LNCS Vol. 1429.

OTERO PÉREZ, C.M. [2001], Method and system for allocating a budget surplus to a task, *Online European Patent Register*, Koninklijke Philips Electronics (applicant), http://register.epoline.org/espacenet/ep/en/srch-reg.htm, Publ. Nr.: WO03044655, Publ. date: 30 May 2003, Prio. Nr.: EP20010204415, Prio. date: 19 November 2001.

OTERO PÉREZ, C.M., R.J. BRIL, AND E.F.M. STEFFENS [2001], Conditionally guaranteed budgets: design and initial analysis, *Proc. Work-in-Progress (WiP) session of the* 22nd *IEEE Real-Time Systems Symposium (RTSS)*, 49–52, Report YCS 337, University of York, Department of Computer Science, UK.

OTERO PÉREZ, C.M., AND I. NITESCU [2002], Quality of service resource management for consumer terminals: Demonstrating the concepts, *Proc. Work-in-Progress (WiP) session of the* 14th *Euromicro Conf. on Real-Time Systems (ECRTS)*, 29–32, Research report 36/2002, Vienna University of Technology, http://www.vmars.tuwien.ac.at.

OTERO PÉREZ, C.M., L. STEFFENS, P. VAN DER STOK, S. VAN LOO,

A. ALONSO, J.F. RUIZ, R.J. BRIL, AND M. GARCÍA VALLS [2003], QoS-based resource management for ambient intelligence, in: T. Basten, M. Geilen, and H. de Groot (eds.), *Ambient Intelligence: Impact on embedded system design*, Kluwer Academic Publishers, Boston/Dordrecht/London, 159–182.

OTT, M., G. MICHELITSCH, D. REININGER, AND G. WELLING [1998], An architecture for adaptive QoS and its application to multimedia systems design, *Computer Communications* **21**:4, 334–349.

PALENCIA, J.C., AND M. GONZÁLEZ HARBOUR [1998], Schedulability analysis for tasks with static and dynamic offsets, *Proc.* 19*th* *IEEE Real-Time Systems Symposium (RTSS)*, 26–37.

PALENCIA GUTIÉRREZ, J.C., J.J. GUTIÉRREZ GARCÍA, AND M. GONZÁLEZ HARBOUR [1998], Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems, *Proc.* 10*th* *EuroMicro Workshop on Real-Time Systems*, 35–44.

PENG, S. [2001], Complexity scalable video decoding via IDCT data pruning, *Digest of technical papers International Conference on Consumer Electronics (ICCE)*, 74–75.

RAEMDONCK, W. VAN, G. LAFRUIT, E.F.M. STEFFENS, C.M. OTERO PÉREZ, AND R.J. BRIL [2002], Scalable 3D graphics processing in consumer terminals, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 369–372.

RAJKUMAR, R., K. JUVVA, A. MOLANO, AND S. OIKAWA [1998], Resource kernels: A resource-centric approach to real-time and multimedia systems, *Proc. SPIE Vol. 3310, Conference on Multimedia Computing and Networking*, 150–164.

REDELL, O. [2003], *Response time analysis for implementation of dsitributed control systems*, Phd thesis, Royal Institute of Technology (KTH), S-100 44 Stockholm, Sweden, http://www.md.kth.se/RTC/Papers/Redell-thesis.pdf.

REDELL, O., AND M. SANFRIDSON [2002], Exact best-case response time analysis of fixed priority scheduled tasks, *Proc.* 14*th* *Euromicro Conf. on Real Time Systems (ECRTS)*, 165–172.

SABATA, B., S. CHATTERJEE, AND J. SYDIR [1998], Dynamic adaptation of video for transmission under resource constraints, *Proc. IEEE International Conference on Image Processing (ICIP), Vol. 3*, 22 – 26.

SHA, L., M.H. KLEIN, AND J.B. GOODENOUGH [1991], Rate monotonic analysis for real-time systems, in: A.M. van Tilborg and G.M. Koob (eds.), *Foundations of real-time computing – Scheduling and Resource Management*, Kluwer Academic Publishers (KAP), 129 – 155.

SHA, L., J. LEHOCZKY, AND R. RAJKUMAR [1986], Solutions for some practical problems in prioritized preemptive scheduling, *Proc.* 7th *IEEE Real-Time Systems Symposium (RTSS)*, 181–191.

SHA, L., R. RAJKUMAR, AND J.P. LEHOCZKY [1990], Priority inheritance protocols: an approach to real-time synchronisation, *IEEE Transactions on Computers* **39**:9, 1175–1185.

SHA, L., R. RAJKUMAR, J.P. LEHOCZKY, AND K. RAMAMRITHAM [1989], Mode change protocols for priority-driven pre-emptive scheduling, *The Journal of Real-Time Systems* **1**:3, 224–264.

SIMONSON, J., AND J.H. PATEL [1995], Use of preferred preemption points in cache-based real-time systems, *Proc. IEEE International Computer Performance and Dependability Symposium (IPDS)*, 316–325.

SJÖDIN, M. [2000], *Predictable high-speed communications for distributed real-time systems*, Ph.D. thesis, Uppsala University, P.O. Box 325, SE-751 05 Uppsala, Sweden, http://user.it.uu.se/ mic/thesis.shmtl.

SJÖDIN, M., AND H. HANSSON [1998], Improved response-time analysis calculations, *Proc.* 19th *IEEE Real-Time Systems Symposium (RTSS)*, 399–408.

SLAVENBURG, G., S. RATHNAM, AND H. DIJKSTRA [1996], The TriMedia TM-1 PCI VLIW mediaprocessor, *Proc.* 8th *Symposium on High Performance Chips, Hot Chips 8*, 171–177.

SPRUNT, B., L. SHA, AND J.P. LEHOCZKY [1989], Aperiodic task scheduling for hard real-time systems, *The Journal of Real-Time Systems* **1**, 27–60.

SPURI, M. [1996], *Analysis of deadline scheduled real-time systems*, Technical Report 2772, Institut National de Recherche et Informatique et en Automatique (INRIA), France.

STEFFENS, E.F.M. [2002], QoS, consumer electronics, and real-time: Challenges and opportunities, http://www.idt.mdh.se/ecrts02/keynote-steffens.pdf.

THUEL, S.R., AND J.P. LEHOCZKY [1994], Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing, *Proc.* 15th *IEEE Real-Time Systems Symposium (RTSS)*, 22–33.

TINDELL, K.W. [1992], *An extendible approach for analysing fixed priority hard real-time tasks*, Report YCS 189, Department of Computer Science, University of York.

TINDELL, K.W. [1994], *Adding time-offsets to schedulability analysis*, Report YCS 221, Department of Computer Science, University of York.

TINDELL, K., AND A. ALONSO [1996], *A very simple protocol for mode changes in priority preemptive systems*, Technical report, Universidad Politécnica de Madrid.

TINDELL, K.W., A. BURNS, AND A.J. WELLINGS [1992], Mode changes in priority pre-emptively scheduled systems, *Proc.* 13th *IEEE Real-Time Systems Symposium (RTSS)*, 100–109.

TINDELL, K.W., A. BURNS, AND A.J. WELLINGS [1994], An extendible approach for analyzing fixed priority hard real-time tasks, *Real-Time Systems* **6**:2, 133–151.

UYKAN, Z. [1997], Hierarchical control and multimedia, in: H. Hyötyniemi and H. Koiva (eds.), *Multimedia applications in industrial automation - Collected papers of the Spring 1997 postgraduate seminar*, Helsinki University of Technology, 91–114, Report 106.

VUORIMAA, P. [2000], A digital television architecture, *Proc. IEEE International Conference on Multimedia and Expo (ICME), Vol. 3*, 1411–1414.

WUBBEN, R., AND C. HENTSCHEL [2003], Using shot-change information to prevent an overload of platform resources, *Proc. SPIE Volume 5150, International Conference on Visual Communcations and Image Processing (VCIP)*, 240–250.

WÜST, C.C., L. STEFFENS, R.J. BRIL, AND W.F.J. VERHAEGH [2004], QoS control strategies for high-quality video processing, *Proc.* 16th *Euromicro Conf. on Real-Time Systems (ECRTS)*, 3–12.

WÜST, C., AND W. VERHAEGH [2001], Quality level control for scalable media processing applications having fixed CPU budgets, *Proc.* 2nd *Philips Workshop on Scheduling and Resource Management (SCHARM)*, 29–39, http://www.extra.research.philips.com/publ/nl-ms/r-21.339.pdf.

WÜST, C.C., AND W.F.J. VERHAEGH [2002], Dynamic control of scalable media processing applications, *Proc.* 1st *Philips Symposium on Intelligent Algorithms (SOIA)*, 119–131, http://www.extra.research.philips.com/publ/nl-ms/r-21.959.pdf.

WÜST, C., AND W. VERHAEGH [2004], Quality level control for scalable media processing applications having fixed CPU budgets, *Journal of Scheduling* **7**:2, 105 – 117.

YANAGIHARA, H., M. SUGANO, A. YONEYAMA, AND Y. NAKAJIMA [2000], Scalable video decoder and its application to multi-channel multicast system, *IEEE Transactions on Consumer Electronics (TCE)* **46**:3, 866–871.

ZHONG, Z., AND Y. CHEN [2001], Scaling in MPEG-2 decoding loop with mixed processing, *Digest of technical papers International Conference on Consumer Electronics (ICCE)*, 76–77.

ZHONG, Z., Y. CHEN, AND T.-H. LAN [2002], Signal adaptive processing in MPEG-2 decoders with embedded re-sizing for interlaced video, *Proc. SPIE Volume 4671, International Conference on Visual Communcations and Image Processing (VCIP)*, 434–441.

Accomplishments

This appendix contains a list of papers of which the undersigned is an author, and a list of patent publications of which the undersigned is an inventor; status June 30th, 2004. Moreover, the undersigned received personal invitations to present a keynote at the ECRTS[1] in 2002 and to join ARTIST[2], which he both passed to his project members. These results were achieved in a five years period, from April 1st 1999 till June 30th 2004 while the undersigned was employed at Philips Research.

Papers

This section is split in two subsection, corresponding with two completely different topics of the publications. Whereas the first subsection contains papers related with the topic of this thesis, the papers in the second subsection have architectural maintenance as topic. The lists of papers contain a total of 25 accepted papers, of which one has not been published yet.

The lists are given in reverse chronical order.

V-QoS: scalable video algorithms and QoS-based resource management

Published papers:

1. R.J. BRIL, W.F.J. VERHAEGH, AND J.J. LUKKIEN, Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption, *Proc. Work in Progress (WiP) session of the* 16th *Euromicro Conference on Real-Time Systems (ECRTS), Technical Report from the University of Nebraska-Lincoln, Department of Computer Science and Engineering (TR-UNL-CSE-2004-0010)*, pp. 57 – 60, June 2004.
2. C.C. WÜST, L. STEFFENS, R.J. BRIL, AND W.F.J. VERHAEGH, QoS Control Strategies for High-Quality Video Processing, *Proc.* 16th *Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 3 – 12, June 2004.[3].

[1]ECRTS is an acronym of Euromicro Conference on Real-Time Systems.

[2]ARTIST is a project in the Information Society Technologies framework programme 6. ARTIST is an acronym for Advanced Real-Time Systems.

[3]Received a best-paper award (as submitted)

3. R.J. BRIL, E.F.M. STEFFENS, AND W.F.J. VERHAEGH, Best-case response times and jitter analysis of real-time tasks, *Journal of Scheduling*, 7(2): 133–147, March 2004.[4]
4. C. HENTSCHEL, R.J. BRIL, Y. CHEN, R. BRASPENNING, AND T.-H. LAN, Video Quality-of-Service for consumer terminals - a novel system for programmable components, *IEEE Transactions on Consumer Electronics*, 49(4): 1367–1377, November 2003.
5. R.J. BRIL, W.F.J. VERHAEGH, AND E.-J.D. POL, Initial values for on-line response time calculations, *Proc.* 15^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 13–22, July 2003.[5]
6. C.M. Otero Pérez, L. Steffens, P. van der Stok, S. van Loo, A. Alonso, J.F. Ruiz, R.J. Bril, and M. García-Valls, QoS-based resource management for ambient intelligence, in: T. Basten, M. Geilen, and H. de Groot (eds.), *Ambient Intelligence: Impact on embedded system design*, Kluwer Academic Publishers, Boston/Dordrecht/London, pp. 159–182, 2003.
7. C. HENTSCHEL, R.J. BRIL, AND Y. CHEN, Video Quality-of-Service for Multimedia Consumer Terminals - An Open and Flexible System Approach (invited paper), *Proc. International Conference on Communications, Circuits and Systems (ICCCS), Vol. 1: Communication Theory and Systems*, pp. 659–663, June 2002.
8. C. HENTSCHEL, R.J. BRIL AND Y. CHEN, Systemkonzept für Multimedia Consumer Terminals mit Dynamisch skalierbaren Algorithmen, *Proc. 20. Jahrestagung der Ferneseh- und Kinotechnischen Gesellschaft (FKTG)*, June 2002 (in German).
9. C. HENTSCHEL, R.J. BRIL, Y. CHEN, R. BRASPENNING, AND T.-H. LAN, Video Quality-of-Service for Consumer Terminals - A Novel System for Programmable Components, *Digest of technical papers International Conference on Consumer Electronics (ICCE)*, pp. 28–29, June 2002.
10. W. VAN RAEMDONCK, G. LAFRUIT, E.F.M. STEFFENS, C.M. OTERO PÉREZ, AND R.J. BRIL, Scalable 3D graphics processing in consumer terminals, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, pp. 369–372, August 2002.
11. C.M. OTERO PÉREZ, R.J. BRIL, AND E.F.M. STEFFENS, Conditionally guaranteed budgets: design and initial analysis, *Proc. Work-in-Progress (WiP) session of the* 22^{nd} *IEEE Real-Time Systems Symposium (RTSS)*, Re-

[4]This paper is based on [Bril, Steffens & Verhaegh, 2001].

[5]This paper is based on [Bril, Pol & Verhaegh, 2002].

port YCS 337, University of York, Department of Computer Science, UK, pp. 49–52, December 2001.

12. R.J. BRIL, C. HENTSCHEL, E.F.M. STEFFENS, M. GABRANI, G.C. VAN LOO, AND J.H.A. GELISSEN, Multimedia QoS in consumer terminals (invited lecture), *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 332–343, September 2001.
13. C. HENTSCHEL, R.J. BRIL AND Y. CHEN, How to add video applications to fully loaded consumer terminals, *Handout at the International Broadcasting Convent (IBC)*, September 2001.
14. R.J. BRIL, E.F.M. STEFFENS, G.C. VAN LOO, M. GABRANI, AND C. HENTSCHEL, Dynamic behavior of consumer multimedia terminals: System aspects, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, pp. 597–600, August 2001.
15. M. GABRANI, C. HENTSCHEL, L. STEFFENS, AND R.J. BRIL, Dynamic behavior of consumer multimedia terminals: Video processing aspects, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1220–1223, August 2001.
16. R.J. BRIL AND E.F.M. STEFFENS, User focus in consumer terminals and conditionally guaranteed budgets, *Proc.* 9^{th} *International Workshop on Quality of Service (IWQoS)*, Lecture Notes in Computer Science (LNCS) 2092, Springer-Verlag, pp. 107–120, June 2001.
17. C. HENTSCHEL, M. GABRANI, K. VAN ZON, R.J. BRIL, AND L. STEFFENS, Scalable Video Algorithms and Quality-of-Service Resource Management for Consumer Terminals, *Digest of technical papers International Conference on Consumer Electronics (ICCE)*, pp. 338-339, June 2001.
18. R.J. BRIL, M. GABRANI, C. HENTSCHEL, G.C. VAN LOO, AND E.F.M. STEFFENS, QoS for consumer terminals and its support for product families, *Proc. International Conference on Media Futures (ICMF)*, pp. 299–302, May 2001.
19. C. HENTSCHEL, R.J. BRIL, M. GABRANI, L. STEFFENS, K. VAN ZON, AND S. VAN LOO, Scalable Video Algorithms and Dynamic Resource Management for Consumer Terminals (invited paper), *Proc. International Conference on Media Futures (ICMF)*, pp. 193–196", May 2001.

Architectural Maintenance

Published papers:

20. R.J. BRIL, A. POSTMA, AND R.L. KRIKHAAR, Embedding architectural support in industry, *Proc. International Conference on Software Mainte-*

nance (ICSM), pp. 348–357, September 2003.[6]

21. R.J. BRIL AND A. POSTMA, An architectural connectivity metric and its support for incremental re-architecting of large legacy systems, *Proc. 9th International Workshop on Program Comprehension (IWPC)*, pp. 269–280, May 2001.
22. R.J. BRIL AND A. POSTMA, A new architectural metric and its visualisation to support incremental re-architecting of large legacy systems, *Proc. 4th International Software Architecture Workshop (ISAW)*, pp. 17–26, June 2000. Proceedings available as: http://www.extra.research.philips.com/SAE/papers/W01_ISAW-4_notes_1.1.pdf
23. R.J. BRIL, L.M.G. FEIJS, A. GLAS, R.L. KRIKHAAR, AND M.R.M. WINTER, Hiding expressed using Relation Algebra with Multi-relations - oblique lifting and lowering for unbalanced systems -, *Proc. 4th European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 33–43, March 2000.
24. R.J. BRIL, L.M.G. FEIJS, A. GLAS, R.L. KRIKHAAR, AND M.R.M. WINTER, Maintaining a legacy: towards support at the architectural level, *Journal of Software Maintenance (JSM)*, 12(3): 143–170, May/June 2000.

Accepted papers:

i. R.J. BRIL, A. POSTMA, AND R.L. KRIKHAAR, Architectural support in industry: A reflection using C-POSH, *Accepted for publication by the Journal of Software Maintenance and Evolution: Research and Practice (invited paper)*, April 2004.

Patents

Table 10.1 provides an overview of the accomplishments of the undersigned over the past five years from an intellectual property perspective. A list of patent publications is given below in reverse chronical order of (earliest) priority number and date (see the Online European Patent Register: http://register.epoline.org/espacenet/ep/en/srch-reg.htm).

1. C. HENTSCHEL AND R.J. BRIL (inventors), *Method and system for allocating shared resources between applications*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO2004027613, Publ. date: 1 April 2004, Prio. Nr.: EP20020078894, Prio. date: 20 September 2002.

[6]Received a best-paper award (as submitted).

Table 10.1. Accomplishments of the undersigned as (co-) inventor over the past five years. The first row summarizes the spin-offs of the work described in this thesis, and the second row provides the results achieved within the V-QoS program. The *published patents* are actually registered as a patent. The *pending patents* are filed and already received a priority number, but have not been published yet. The *pending invention disclosures* have been submitted to Philips' intellectual property department, but have not been handled yet. The '*' denotes accomplishments during secondment from Philips Research to the TU/e.

	published patents	pending patents	pending invention disclosures	total
PhD	3	7*	1*	11
V-QoS	10	1	3*	14
total	13	8	4	25

2. M. Gabrani, C. Hentschel, R.J. Bril, and E.F.M. Steffens (inventors), *Processing a media signal in a media system to prevent overload*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO03103296, Publ. date: 11 December 2003, Prio. Nr.: EP20020077135, Prio. date: 30 May 2002.
3. C. Hentschel, M. Gabrani, R.J. Bril, and E.F.M. Steffens (inventors), *Image processing method and system to increase perceived visual quality in case of lack of image data*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO03063507, Publ. date: 31 June 2003, Prio. Nr.: EP20020075281, Prio. date: 23 January 2002.
4. C. Hentschel, M. Gabrani, R.J. Bril, E.F.M. Steffens, and C.C.A.M. van Zon (inventors), *Processing a media signal on a media system*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO03050758, Publ. date: 19 June 2003, Prio. Nr.: EP20010204857, Prio. date: 12 December 2001.
5. M. Gabrani, C. Hentschel, E.F.M. Steffens, and R.J. Bril (inventors), *A method to assist in the predictability of open and flexible systems using video analysis*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO03036941, Publ. date: 01 May 2003, Prio. Nr.: EP20010204080, Prio date: 25 October 2001.
6. G. Lafruit, R.J. Bril, and E.F.M. Steffens (inventors), *A method for operating a real-time multimedia terminal in a QoS manner*, Interuniversity MicroElectronics Center (IMEC) and Koninklijke Philips Electronics N.V. (applicants), Publ. Nr.: EP1296243, Publ. date: 26 March 2003, Prio. Nr.:

US20010325268P, Prio. date: 25 September 2001.

7. M. GABRANI, C. HENTSCHEL, C.C.A.M. VAN ZON, E.F.M. STEFFENS, AND R.J. BRIL (inventors), *Method of running a media application and a media system with job control*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO2003007134, Publ. date: 23 January 2003, Prio. Nr.: EP20010117107, Prio date: 13 July 2001.
8. E.F.M. STEFFENS AND R.J. BRIL (inventors), *A method of and system for assessing progress of a task*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO02099626, Publ. date: 12 December 2002, Prio. Nr.: EP20010202128, Prio. date: 05 June 2001.
9. C.M. OTERO PÉREZ, E.F.M. STEFFENS, AND R.J. BRIL (inventors), *Method of and system for withdrawing budget from a blocking task*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO02071218, Publ. date: 12 September 2002, Prio. Nr.: EP20010200810, Prio. date: 05 March 2001.
10. C. HENTSCHEL, C.C.A.M. VAN ZON, S. PENG, Z. ZHONG, M. GABRANI, E.F.M. STEFFENS, AND R.J. BRIL (inventors), *Method of and system for running an algorithm*, Publ. Nr.: US20020129080, Publ. date: 12 September 2002, Prio. date: 11 January 2001.
11. R.J. BRIL AND E.F.M. STEFFENS (inventors), *Method of and system for determining a best-case response time of a periodic task*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO0239256, Publ. date: 16 May 2002, Prio. Nr.: EP20000203904, Prio. date: 09 November 2000.
12. R.J. BRIL, E.F.M. STEFFENS, C. HENTSCHEL, M. GABRANI, C.C.A.M. VAN ZON (inventors), *A method and a system for allocation of a budget to a task*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO0237275, Publ. date: 10 May 2002, Prio. Nr.: EP20000203876, Prio. date: 06 November 2000.
13. C. HENTSCHEL, S. PENG, C.C.A.M. VAN ZON, M. GABRANI, E.F.M. STEFFENS, AND R.J. BRIL (inventors), *Method of running an algorithm and a scalable programmable processing device*, Koninklijke Philips Electronics N.V. (applicant), Publ. Nr.: WO0219095, Publ. date: 07 March 2002, Prio. Nr.: US20000649777, Prio. date: 29 August 2000.

Symbol Index

The numbers refer to the pages of first occurrence.

Schedules

$\sigma(t)$	task of task set $\mathcal{T}$ assigned to the processor at time t	40
$\sigma_i(t)$	task of task set τ_1 till τ_i assigned to the processor at time t	40

Characteristics of task τ_i and task set $\mathcal{T}$

BC_i	best-case computation time of task τ_i	41
C_i	computation time of task τ_i	44
D_i	(relative) deadline of task τ_i	41
φ_i	phasing of task τ_i	41
φ	phasing of task set $\mathcal{T}$	41
T_i	period of task τ_i	41
WC_i	worst-case computation time of task τ_i	41

Characteristics of subjob l of task τ_i

$BC_{i,l}$	best-case computation time of subjob l of task τ_i	164
$WC_{i,l}$	worst-case computation time of subjob l of task τ_i	164

Times and length of intervals of job k of task τ_i

a_{ik}	activation (release or request) time	41
b_{ik}	begin (or start) time	41
d_{ik}	absolute deadline	41
e_{ik}	length of the execution interval	41
f_{ik}	finalization (or completion) time	41
r_{ik}	length of the response interval	41
s_{ik}	length of the start interval	41

Best-case and worst-case times of task τ_i

BE_i	best-case execution time	43
BO_i	best-case occupied time	83
BR_i	best-case response time	43
BS_i	best-case start time	43
WE_i	worst-case execution time	42
WO_i	worst-case occupied time	59
WR_i	worst-case response time	42
WS_i	worst-case start time	42

Blocking and specialized worst-case times of task τ_i

WB_i	worst-case blocking time	164
WO_i^{P}	worst-case occupied time under FPPS	164
WR_i^{D}	worst-case response time under FPDS	164
WR_i^{N}	worst-case response time under FPNS	164
WR_i^{P}	worst-case response time under FPPS	164

Initial values for calculating response times and occupied times of task τ_i

ι_i^{A}	alternative initial value for real-time analysis	86
ι_i^{AB}	alternative initial value for best-case analysis	69
ι_i^{AW}	alternative initial value for worst-case analysis	69
ι_i^{CB}	combined initial value to calculate BR_i	87
ι_i^{CW}	combined initial value for worst-case analysis	69
ι_i^{SB}	standard initial value to calculate BR_i	86
ι_i^{SW}	standard initial value for worst-case analysis	68

Jitter of task τ_i

AJ_i	activation (or release) jitter of task τ_i	159
FJ_i	finalization (or completion) jitter of task τ_i	158

Advancements of task τ_i

BA_i	best-case advancement of task τ_i	130
WA_i	worst-case advancement of task τ_i	130

Utilization factors

BU_i^τ	best-case utilization factor of task τ_i	44
BU_i	best-case cumulative utilization of tasks τ_1 till τ_i	44
BU	best-case (processor) utilization factor of $\mathcal{T}$	44
U_i^τ	utilization factor of task τ_i	44
U_i	cumulative utilization factor of tasks τ_1 till τ_i	44
U	(processor) utilization factor of $\mathcal{T}$	44
WU_i^τ	worst-case utilization factor of task τ_i	44
WU_i	worst-case cumulative utilization of tasks τ_1 till τ_i	44
WU	worst-case (processor) utilization factor of $\mathcal{T}$	44

Characteristics of budget β_i and budget set $\mathcal{B}$

Most of the symbols for tasks and sets of tasks also apply to budgets and sets of budgets, respectively. Below, an overview of those symbols is given that have been described explicitly in the budget model.

B_i	budget of budget β_i	108
D_i	(relative) deadline of budget β_i	108
φ_i	phasing of budget β_i	108
φ	phasing of budget set $\mathcal{B}$	108
T_i	period of budget β_i	108

Conditionally guaranteed budgets

ΔB_c^{B}	best-case CGB contribution function	145
ΔB_c^{W}	worst-case CGB contribution function	145
ΔB_c	conditionally guaranteed budget of CGB_c	133
ΔB_p	budget margin of AGB_p	133
T_c^{EL}	exact lower bound	142
T_c^{EU}	exact upper bound	135
T_c^{PL}	pessimistic lower bound	142
T_c^{PU}	pessimistic upper bound	142

Author Index

Subject Index

Samenvatting

In dit proefschrift behandelen we een planningsprobleem dat haar oorsprong vindt in het kosteneffectief verwerken van verschillende media door software in consumentenapparaten, zoals digitale televisies.

De laatste jaren zijn er trends gaande van analoge naar digitale systemen, en van verwerking van digitale signalen door specifieke, toepassingsgerichte hardware naar verwerking door software. Voor de verwerking van digitale media door software wordt gebruik gemaakt van krachtige programmeerbare processoren. Om te kunnen wedijveren met bestaande oplossingen is het van belang dat deze programeerbare hardware zeer kosteneffectief wordt gebruikt. Daarnaast dienen de bestaande eigenschappen van deze consumenten apparaten, zoals robuustheid, stabiliteit, en voorspelbaarheid, behouden te blijven als er software wordt gebruikt. Verder geldt dat er gelijktijdig meerdere media stromen door een consumenten apparaat verwerkt moeten kunnen worden. Deze uitdaging is binnen de onderzoekslaboratoria van Philips aangegaan in het zogenoemde Video-Quality-of-Service programma, en het werk dat in dit proefschrift beschreven wordt is binnen dat programma ontstaan. De binnen dat programma gekozen aanpak is gebaseerd op schaalbare algoritmen voor de verwerking van media, budgetten voor die algoritmen, en software dat de instelling van die algoritmen en de grootte van de budgetten aanpast tijdens de verwerking van de media. Ten behoeve van het kosteneffectief gebruik van de programmeerbare processoren zijn de budgetten krap bemeten.

Dit proefschrift geeft een uitvoerige beschrijving van die aanpak, en van een model van een apparaat dat de haalbaarheid van die aanpak aantoont. Vervolgens laten we zien dat die aanpak leidt tot een probleem wanneer er gelijktijdig meerdere stromen worden verwerkt die verschillende relatieve relevanties hebben voor de gebruiker van het apparaat. Om dit probleem op te lossen stellen we het nieuwe concept van voorwaardelijk gegarandeerde budgetten voor, en beschrijven we hoe dat concept kan worden gerealiseerd. De technieken voor het analyseren van het planningprobleem voor budgetten zijn gebaseerd op bestaande technieken voor slechtste-gevals-analyse voor periodieke real-time taken. We breiden die bestaande technieken uit met technieken voor beste-gevals-analyse zodat we apparaten die gebruik maken van dit nieuwe type budget kunnen analyseren.

Curriculum Vitae

Reinder J. Bril was born in Uithuizen (Gr.), the Netherlands, on August 2^{nd}, 1958. He married Carmen V.J. Koning in December 1993, and has three lovely children (two daughters and a son): Joëlla V. (1994), Marijn J. (1997), and Wander J. (2001).

He received his entire training in the Netherlands. In 1976, he finished secondary school (so-called ‘Atheneum-B’) at the *Baudartius College* in Zutphen, and started at the *University of Twente* (UT) in Enschede. He performed his practical work at the *Centre for Educational Technology* in Tel Aviv, Israel. He became fully qualified to teach mathematics and mechanical engineering and received a B.Sc. and a M.Sc. (both with honors) from the Department of Electrical Engineering of the UT in January 1981 and January 1984, respectively.

Based on invitation, he started his professional career at the *Delft University of Technology* in the Department of Electrical Engineering. Since May 1985, he has been with *Koninklijke Philips Electronics N.V.*. He has worked in both Philips Research as well as Philips Business Units, on various topics, including fault tolerance, formal specifications, and software architecture analysis, in different application domains, and in a variety of roles, such as software engineer, software architect, senior scientist, project leader, and consultant. In April 1999, he returned to Philips Research in Eindhoven, to work in the area of Quality-of-Service for consumer devices, with a focus on dynamic resource management in receivers in broadcast environments (such as digital TV sets and set-top boxes).

In May 2003, he was sent on secondment for a period of a year by Philips to the *Technische Universiteit Eindhoven* (TU/e), Department of Mathematics and Computer Science, Group System Architecture and Networking with the prime goal to finish his Ph.D. and this thesis. Assuming successful, he will make a transfer back to the academic world after 19 years in industry.

“ ‘Tut, tut,’ hernam de professor. ‘Welk een beuzelpraat! Ik begin te geloven dat ge toch een student zijt.’ ”

Marten Toonder, *De killers*, [5340], 1964.

Titles in the IPA Dissertation Series

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unifi cation-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfi ability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratifi ed approach to the verifi cation of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verifi cation of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specifi cation Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space*