Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management

Linköping Studies in Science and Technology
Dissertations. No. 1559

Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management

by

Sergiu Rafiliu

Linköping University

Department of Computer and Information Science
Linköping Univeristy
SE–581 83 Linköping, Sweden

Linköping 2013

ISBN 978-91-7519-471-4 ISSN 0345-7524
Printed By LiU-Tryck 2013
URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-98721

To Nico

Abstract

TODAY'S embedded distributed real-time systems, are exposed to large variations in resource usage due to complex software applications, sophisticated hardware platforms, and the impact of their run-time environment. As efficiency becomes more important, the applications running on these systems are extended with on-line resource managers whose job is to adapt the system in the face of such variations. Distributed systems are often heterogeneous, meaning that the hardware platform consists of computing nodes with different performance, operating systems, and scheduling policies, linked through one or more networks using different protocols.

In this thesis we explore whether resource managers used in such distributed embedded systems are stable, meaning that the system's resource usage is controlled under all possible run-time scenarios. Stability implies a bounded worst-case behavior of the system and can be linked with classic real-time systems' properties such as bounded response times for the software applications. In the case of distributed systems, the stability problem is particularly hard because software applications distributed over the different resources generate complex, cyclic dependencies between the resources, that need to be taken into account. In this thesis we develop a detailed mathematical model of an adaptive, distributed real-time system and we derive conditions that, if satisfied, guarantee its stability.

Popupärvetenskaplig sammanfattning

Vi är omgivna av ett ständigt ökande antal inbyggda datorsystem. De finns exempelvis i våra bilar, telefoner, fotokameror och tvättmaskiner. Vår förväntning är att dessa produkter ska vara säkra, effektiva samt hålla hög kvalitet på de tjänster de tillhandahåller. Detta innebär bland annat att de datorsystem som finns inbyggda i våra bilar inte får leda till att passagerare skadas eller i värsta fall mister sina liv. Av en telefon förväntar vi oss lång batteritid även om vi använder dessa dagligen för att prata och skicka SMS, lyssna på musik, ta bilder, läsa nyheter och skicka e-post. Vi förväntar oss samtidigt att de tjänster och funktioner som finns i våra mobiltelefoner håller hög kvalitet. Alla våra krav och förväntningar kan endast uppfyllas om systemens inbyggda resurser används på ett effektivt sätt. Exempel på sådana resurser är processorer, minnen, batterier och kommunikationsmedier.

I denna avhandling beskrivs hur resurser i inbyggda datorsystem bör användas och kontrolleras för att optimera dess tjänster. Vidare tas hänsyn till att ett modernt system har stora variationer i resursanvändning. Detta beror delvis på att det mångfald av tjänster som finns i dagens moderna datorsystem i allra högsta grad består av komplex programvara. Detta beror även på den komplicerade och sofistikerade hårdvara som behövs för att stödja all inbyggd programvara samt påverkan från systemets omgivning.

För att kunna förlita oss på dessa inbyggda system är det nödvändigt att studera huruvida dess resurshanterare är stabila. Denna avhandling presenterar villkor för stabilitet vilket innebär att resursernas användning är kontrollerbar under alla tänkbara scenarier. Vi visar att detta begrepp kan överbryggas till realtidsegenskaper såsom begränsade responstider för all inbyggd programvara. Stabilitetskravet är svårt att hantera i synnerhet för distribuerade system eftersom programvaran är utspridd på olika systemresurser. I denna avhandling visar vi att detta leder till cykliska resursberoenden samt tar hänsyn till dessa komplicerade egenskaper för att härleda villkor för stabilitet.

Acknowledgments

THERE are many people who have contributed, in a way or another, to the development of this thesis. First I would like to thank my thesis advisers prof. Petru Eleş and prof. Zebo Peng for the time that they have invested into my education, the inspiring and sometimes energetic meetings, and the encouragement and support that they have offered during my years as a PhD student.

I would also like to thank prof. Michael Lemmon and the members of his research group at the University of Notre Dame (USA), Department of Electrical Engineering, for the insightful discussion we had during my time there.

Many thanks go to my colleagues at the Department of Computer and Information Science (IDA) for the excellent working environment, especially to Eva Pelayo Danils, Gunilla Mellheden, and Anne Moe for making the administrative work painless.

My appreciation and gratitude go to the present and former members of the Embedded Systems Laboratory (ESLAB) for the many discussions, lunches and companionship that they have offered. I thank early members of ESLAB: Soheil Samii, Traian Pop, Alexandru Andrei, Zhiyuan He, and Jakob Rosén for helping me adjust to the PhD life and for helping me grow as a researcher. I also thank Soheil Samii and Bogdan Tanasă for their willingness to help with detailed discussions regarding my work.

Last, I would like to thank my family (Letiţia, Aurel, and Camelia) for their support during these period of time. This thesis is dedicated to my wife, Nicoleta, to whom I express my deepest gratitude for her infinite love, patience, and understanding.

Sergiu Rafiliu
Linköping, 17 Dec. 2013

Contents

List of Figures

List of Tables

1 Introduction

THE TOPIC of this thesis is stability of adaptive real-time systems. The main contributions are a detailed mathematical modeling of such systems and conditions that these systems must satisfy in order to be stable. In this chapter we shall introduce and motivate this research topic as well as give a general description about adaptive real-time systems and the problems they face. We shall present here our contributions and an overview of the organization of this thesis.

1.1 Motivation

Today's embedded systems, together with the real-time applications running on them, have a high level of complexity [Kop11]. Moreover, such systems very often are exposed to varying resource demand due to e.g. variations in the execution times of the software tasks in the system. When these variations are large and system efficiency is required, on-line resource managers may be deployed to control the system's resource usage. Among the goals of such a resource manager is to maximize the resource usage while minimizing the amount of time the system spends in overload situations.

Examples of such systems are found in, for example, multimedia devices [Lee99], automotive applications [Bur98], or control applications [Set96]. The resources might represent the different computation nodes and the communication infrastructure in the system and the variations in resource demand might, for example, originate from sensors that provide different inputs to software tasks which leads to variations in their execution times. A resource manager will monitor the resources and control the behavior of the system such that it keeps its real-time properties e.g. all computations finish and all messages are sent in finite amounts of time.

A key question to ask for such systems is whether the deployed resource managers are safe, meaning that the resource demand is bounded under all possible runtime scenarios. This notion of safety can be linked with the notion of stability of control systems. In control applications, a controller controls the state of a plant towards a desired stationary point. The system is stable if the plant's state remains within a bounded distance from the stationary point under all possible scenarios [Gla00, Son98]. By modeling the real-time system as the plant and the resource manager as the controller, one may be able to reason about the stability of the system.

1.2 Problem Description

In this work we describe adaptive distributed real-time systems as being composed of three elements:

1. *hardware platform*, which comprises the resources of the system,
2. *software applications*, captured as acyclic task graphs, and
3. *resource manager*, which adapts the system according to changing resource demand.

Resources in the system are all the hardware components which are accessed by the software application tasks. Examples of such resource are CPUs, accessed via the task schedulers built inside their operating systems, and communication links scheduled according to their protocols. In this thesis we can handle any non-idling scheduling policy, thus, covering a large class of resource schedulers currently used [But97].

The software applications running on the system are seen as a set of acyclic task graphs [Cof72, Cas88], each task representing a piece of code that takes a number of inputs and produces a number of outputs. The links in the task graph represent the input-output dependencies between the tasks. As the system runs, it repetitively releases instances of the task graphs with new inputs. Each task occupies only one resource and its resource usage refers to the amount of time that the resource is held executing the code associated with the task. With regards to communication links, a task represents the message that needs to be sent across it.

The resource manager changes parameters of the system in order to adapt it the to changing resource demand. The goal of the adaptation is to improve system performance in average-case scenarios while keeping the system stable during its worst-case behavior. In this thesis we handle adaptations through:

- changes in the rate at which task graph instances get released,
- changes of resource capacity,
- admission/dropping of task graph instances, and
- changes in task execution times.

Changes in task graph rates are used in applications such as web servers [Hen04] and teleconferencing systems [Gho03] where they provide differentiation in quality-of-service levels to end users. Examples of changes of resource capacity are dynamic voltage and frequency scaling and their application is typical for thermal aware and low power embedded systems [Mar07, Bao09] and wireless communication [Chi06]. Admission control/job dropping is used in applications such as web servers [Abd03] where the system, at times, is subjected to very large numbers of incoming requests that cannot be serviced in a timely manner. Adaptation through changes in task execution time is done in systems supporting imprecise computations, such as real-time database servers [Ami06].

1.3 Problem Formulation

The amount of time a task needs to execute before it finishes varies from instance to instance and depends on the inputs given to the task (influencing e.g. what branch of the task's code gets chosen), the state of the resource (state of CPU pipeline and cache memories), the scheduler, etc. and, in general, it cannot be determined precisely. The goal of the resource manager is to adapt the system to such variations in order to improve system performance, but also to avoid situations when task instances queue up in an unbounded way, without being executed. In such situations the system looses its real-time properties and may become hazardous to the environment in which it is deployed.

Our goal is to model and analyze the behavior of the system and resource manager in order to determine conditions under which the whole system is guaranteed to be stable. By stability we mean that all accumulations of task instances are bounded.

1.4 Contributions

In this work we consider a distributed real-time platform formed of a number of resources (processors, buses, etc.) and applications seen as a set of task graphs mapped across the different resources. We assume that the task graphs can release jobs at variable rates and we require knowledge of the intervals in which execution times of jobs of tasks vary (with respect to communication resources, tasks represent messages and jobs correspond to message instances). We allow these jobs to be scheduled on their resources using any kind of non-idling scheduling policy. We also allow that the schedulers are heterogeneous (different schedulers for different resources). Before being scheduled for execution, jobs accumulate in queues, one for each tasks in the system. We consider that the system possesses a resource manager (also distributed across the different resources) whose job is to adjust task rates subject to the variation in job execution times.

We develop criteria that, if satisfied by the system and the resource manager, render the adaptive real-time system stable under any resource demand pattern (execution time variation pattern). Stability implies bounded queues of jobs, for all tasks in the system, and can be linked with bounded worst-case response times for jobs of tasks and bounded throughput.

With the criteria developed in this work we guarantee the stability of the system in all possible cases, meaning that the proposed framework is suitable even for adaptive hard real-time systems.

Before discussing stability, we go through a somewhat involving modeling phase where we develop a detailed, non-linear model of our system (Chapter 4). Our model tracks the evolution in time of the accumulation of execution time on each resource. The accumulation of execution time is the sum of execution times of the queued jobs of all tasks running on a resource. From this model we build a worst-case evolution model that we use when deriving our stability results.

For modeling purposes we need to overcome several challenges. First we need to recognize the parameters whose behavior can be modeled. The state of the system should be formed of the queue sizes of the queues of all tasks in the system. However, since modeling their evolution is not possible, as it depends on the schedulers used, we need to replace this with a more general type of information that describes in an aggregated fashion the queue sizes. We identify this information to be the accumulation of execution times on each resource.

Second, we need to build the model of the system describing the evolution in time of the accumulations of execution times on each resource. Third, we need to determine what is the worst-case behavior of the system. The fourth and final challenge that we face is due to the distributed nature of the system since the accumulation of execution times flows between resources and, thus, we may experience very rich behavior patterns that need to be accounted for when determining the worst-case behavior of the system. We illustrate the sometimes counter intuitive behavior, with regards to stability, of distributed systems in Section 5.1 with a simple example. Our worst-case behavior model is that of a linear switching system with random switching, where the system evolves linearly according to one of several branches, but may randomly switch to a different one at any time.

Given this worst-case behavior model of the adaptive real-time system we develop three criteria that determine if the system is stable or not. The first two criteria (Sections 5.1 and 5.2) consider the topology and parameters of the system (worst-case execution times and rates, mapping, etc.) and determine if there exist resource managers that can keep the system stable. Although the number of branches in our worst-case behavior model is exponential in the number of resources, and we need to account for all possible switching behavior, we manage to formulate conditions whose complexity is linear in the number of resources in the system.

The last criterion (Section 5.3) describes conditions that a resource manager needs to satisfy in order to keep the system stable. Unlike the previous literature (with the possible exception of [Cuc10]) in this paper we do not present a particular, customized method for stabilizing a real-time system. We do not present a certain algorithm or develop a particular controller (e.g. PID, LQG, or MPV controller). Instead, we present a criterion which describes a whole class of methods that can be used to stabilize the system. Also, in this work, we do not address any performance or quality-of-service metric, since our criterion is independent of the optimality with which a certain resource manager achieves its goals in the setting where it is deployed. The criterion that we propose may be used in the following ways:

1. to determine if an existing resource manager is stable,
2. to help build custom, ad-hoc resource managers which are stable, and
3. to modify existing resource managers to become stable.

In this thesis we aim at building a general theory for adaptive distributed real-time systems which describes, in a unified way, the behavior of a large and diversified class of systems (large in terms of all the scheduling and resource management policies accepted) and which solves the basic problem of ensuring system stability in the face of perturbations. The main contributions of the thesis are:

1. Modeling of the evolution in time of adaptive distributed real-time systems that employ non-idling schedulers for their resources.
2. Determining the worst-case behavior of the system.
3. Determining conditions on the parameters of the system that, if satisfied, guarantee the existence of resource managers which can stabilize the system.
4. Determining a condition on the resource manager that, if satisfied, guarantees that the system will remain stable under all its possible evolution patterns.

1.5 List of Publications

Parts of this thesis are presented in the following publications:

- Sergiu Rafiliu, Petru Eles, Zebo Peng.
  "Low Overhead Dynamic QoS Optimization Under Variable

Task Execution Times." 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2010), Macau SAR, P.R.C., August 23-25, 2010 [Raf10].

- Sergiu Rafiliu, Petru Eles, Zebo Peng.
  "Stability Conditions of On-line Resource Managers for Systems with Execution Time Variations." 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011), Porto Portugal, June 6-8, 2011 [Raf11].

- Sergiu Rafiliu, Petru Eles, Zebo Peng.
  "Stability of Adaptive Feedback-based Resource Managers for Systems with Execution Time Variations." Real-Time Systems Journal, vol. 49, issue 3, 2013 [Raf13].

- Sergiu Rafiliu, Petru Eles, Zebo Peng, and Michael Lemmon.
  "Stability of On-line Resource Managers for Distributed Systems under Execution Time Variations." Under review at the ACM Transactions on Embedded Computing Systems Journal [Raf–].

The following publication is not covered in this thesis but is directly related to the field of distributed real-time systems:

- Soheil Samii, Sergiu Rafiliu, Petru Eles, Zebo Peng.
  "A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems." Design, Automation, and Test in Europe (DATE 2008), Munich, Germany, March 10-14, 2008, pp. 556-561 [Sam08].

1.6 Thesis Overview

This thesis is organized in eight chapters. The presentation in Chapters 3–5 is done on a simplified version of the system in order to keep the discussion concise and to limit the number or mathematical notations and other possible sources of distraction. In Chapters 6–8 we eliminate this limitations in order to make our theory more relevant and useful to our domain.

In **Chapter 2** we introduce the necessary concepts regarding adaptive distributed real-time systems needed as a context for this thesis. Here we present the different types of adaptation mechanisms together with their applications. We also present the state-of-the-art research done in the area and show how our work relates to it.

In **Chapter 3** we present the system architecture that we consider, together with all concepts and mathematical notations needed later in the thesis.

In **Chapter 4** we develop the formal model of our system. We then use this model to determine and develop a model of the worst-case behavior of the system. We end the chapter with an illustrative example and an in-depth informal discussion about the interpretation of the model.

In **Chapter 5** we develop the main results of this thesis, which are the three stability criteria that allow us to determine whether a given system can be stabilized at all, and if it is stable under the control of its resource manager.

In **Chapter 6** we present a discussion on the meaning, features, and limitations of our model and of the derived stability criteria. Here we eliminate the previously imposed limitations on the model and extend our stability criteria to handle more general systems.

In **Chapter 7** we show how to apply our stability criteria on several examples, and we show how these are useful in determining meaningful real-time system properties such as worst-case response times.

In **Chapter 8** we conclude this thesis and we outline several directions of future research that closely relate to the theory developed here.

2 Background and Related Work

THE PURPOSE of this chapter is to review some of the efforts made in order to bring adaptivity to embedded real-time systems. Initially embedded real-time systems were simply designed to meet worst-case guarantees [Liu73]. This approach works fine for applications where safety is critical and cost is no issue. Such systems are called *hard real-time systems*. However, for the majority of systems (e.g. real-time systems embedded in consumer electronics) hard real-time constraints are usually not mandatory while cost becomes the dominant issue [Abe98]. Such systems are often designed for average-case performance and are called *soft real-time systems*. They are often characterized by different modes of operation [Kop11] such that a static optimization, which does not consider the switches between the different modes that can happen at run-time, leads to poor performance and suboptimal resource usage. Also, it is often the case that systems optimized for average-case behavior, have sharp drop-offs in performance at their worst-case [Ce03b]. Adaptivity is desirable in such systems in order to achieve graceful performance degradation.

2.1 Adaptive Real-Time Systems

We can think of adaptive systems as having three features:

1. *performance metric* that is to be optimized at run-time,
2. *constraints* that need to be satisfied for correct system operation, and
3. *actuation mechanisms* that are used by the resource manager to trade-off performance for satisfying the constraints.

The resource manager of an adaptive real-time system, in essence, solves an optimization problem whose goal is to maximize performance (given by the performance metric[1]) subject to satisfying the constraints. It runs periodically in the system and, during each run, it detects the state of the system and decides on the achievable performance under current conditions. The resource manager, then, implements the decision via its actuation mechanisms.

The performance metric that adaptive systems optimize on-line, varies greatly from system to system. The performance metric can be represented as: jitter, delay, or quality-of-control for systems running control applications [Ast97]; quality-of-service functions for systems running multimedia applications [Ng02] or web-services [Abd00]; power consumption for low-power devices [Bha10, Zha06]; etc.

Constraints can also be expressed differently between different systems, however, they usually stem from the need of timeliness in real-time systems and can be represented as schedulability tests [Cer02] or tests regarding deadline violations [Abe04].

[1]In optimization theory literature, this is called *cost function.* In the context of this discussion we prefer the term *performance metric* as we use the word *cost* to denote the actual cost of building the system.

Actuation mechanisms are of great importance as they depend on the capabilities of the system and greatly affect the efficiency of the resource manager. For the theoretical framework that we develop in this thesis, the actuation mechanisms are also important as they need to be modeled. We present actuation mechanisms in more detail in Section 2.2.

In this thesis we focus on building a general theory for adaptive real-time systems. To this end, we focus on the least common denominator that all of these systems feature. Here we do not consider any particular performance metrics of the resource management policy, as these vary greatly between different types of adaptive systems. To a large extent, the constraints that adaptive systems must satisfy also vary greatly, however, all the considered systems have in common their need for timeliness: they all must guarantee that their jobs execute in a timely fashion. In the theory that we develop in this thesis, we focus on a very basic form of timeliness which implies, that every job entering the system must exit it in a finite amount of time. We call systems satisfying this property stable and the main focus of the thesis is to provide conditions that systems must satisfy for stability.

2.2 Types of Actuation Mechanisms

There exists a vast literature regarding resource adaptation, targeting different types of real-time systems and using different adaptation mechanisms. Figure 2.1 illustrates a real-time system and some 'knobs' that can be used for adaptation. Adjustments can be applied to the jobs that enter or are already in the system, to the resources, or to the schedulers related to those resources. We, thus, propose the following four classes of adaptation mechanisms:

1. *Job Flow adaptation:* these are mechanisms that adapt the incoming flow of task instances (jobs) according to the current state of the system,

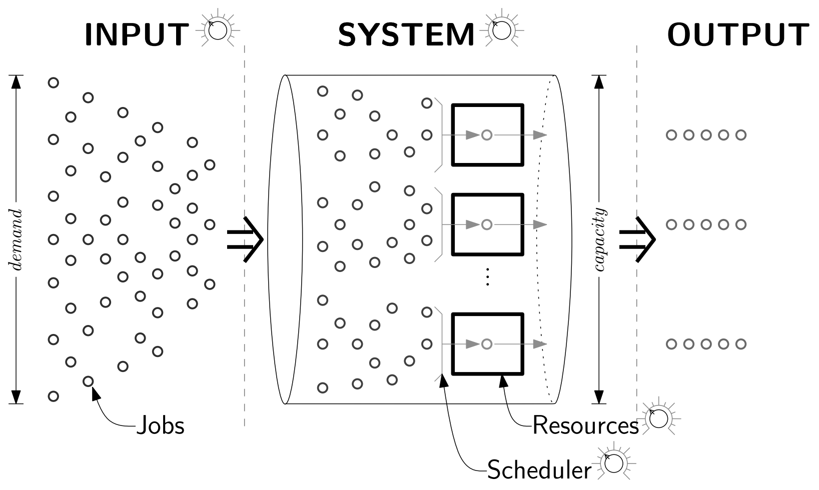


Figure 2.1: A real-time system, together with the possible actuation mechanisms (‘knobs’) that the resource managers may use in order to provide adaptation.

2. *Resource adaptation:* these mechanisms adapt the resource, depending on the current usage pattern required by the application,
3. *Task Mode adaptation:* these are mechanisms that adapt the way the already released jobs get to access the resource,
4. *Schedule adaptation:* these are mechanisms that try to adapt various parameters of the scheduler to improve the performance of the running applications.

Job flow adaptation mechanisms work by shaping the input to the system in order to match its processing capacity. The adaptation is done by changing the software tasks release rates (making jobs appear more slowly) or by admission control (eliminating some of the incoming jobs before they enter the system and use its resources). Papers that fall in this class are [Lee98, But98, But02, But04, Mar07, Lu02, Abd03, Cer02].

Resource adaptation mechanisms work by shaping the resource capacity to meet the incoming demand. This is directly opposite to job flow adaptation mechanisms. Resource adaptation works by changing the capacity of the resources via techniques such as voltage or frequency scaling for processors, or by increasing buss speed for networks. In this category fall methods such as the ones presented in [Mar07].

Task mode adaptation mechanisms work by adapting the behavior of the jobs already admitted into the system to meet the system's resource capacity. This is useful because the resource demand in the system comes both from the incoming jobs (to be released in the future) and from the released, but not yet executed jobs that lie in the system. The adaptation mechanisms are task mode changes, such as in the case of imprecise computation, where computations associated with the jobs are iterative and the number of iterations can be altered to trade-off precision in output for better execution time. An extreme example of this is job dropping where the computation is iterated 0 or 1 times. This class is comprised of methods such as the ones described in [Lee98, Com08, Abd03].

Scheduler adaption mechanisms refer to changes done to the scheduler for improved system response. These adaptation mechanisms are feasible in the case when it is known that the resource demand is generally below the resource capacity. Over short intervals of time, however, resource demand may spike above resource capacity and produce deadline misses, if jobs are not scheduled for execution in the correct order. Works such as [Pa09a, Cuc10, Com08, Liu07, Yao08, Ce03a] fall in this class.

The methods in the first three classes achieve similar goals, they try to keep the utilization of the resources at a prescribed level in the face of varying job execution times and/or arrival patterns. While doing so they also try to maximize one or more quality-of-service or performance metrics. The methods in the fourth class adjust scheduler parameters in an attempt to match the resource demand of each task to a specific share of the capacity of the resource, with the goal of minimizing deadline misses and maximizing various performance or quality-of-service metrics. When discussing the stability of adaptive real-time systems, the issue of handling overload situations arises. By overload we mean that more jobs arrive per unit of time than can be processed by the resource. Only methods from the first three classes can handle these overloads as they adjust the incoming job flow or the capacity of the resource to preserve equilibrium between demand and capacity. The methods from the fourth class are complementary methods that help improve the performance of the system [Liu07].

2.3 Related Work

The aim of this thesis is to build a general theory of modeling and analyzing adaptive real-time systems. As such, there is no direct previous work that we can compare with. In this section we present the state of the art regarding methods for adaptation used in various adaptive real-time systems by following several popular research directions.

Lee et al. proposed the QRAM algorithm in [Raj97, Lee98, Lee99]. The model consists of a number of resources that software applications can use in parallel and a number of abstract quality dimensions. When the algorithm runs, it optimizes the overall quality subject to keeping the resource constraints. It does so by interpreting all quality options in terms of resource demand and it increases its quality along the dimension with the steepest slope, until the resource capacity is fully consumed. The motivation for this work comes from multimedia applications where quality dimensions might be audio and video sampling rates and stream encodings. Sampling rates are an example of job flow adaptation as they affect the stream of incoming frames to be processed, while stream encodings are an example of task mode adaptation as they affect the processing time of a frame. In this line of works, software applications are not given more detailed modeling, except for saying that they may consume multiple resources at the same time. The *QRAM* resource management policy is successful in dealing with abstract notions of quality, and how to determine the optimum balance point between resource demand and capacity. However, little attention is given to the actuation mechanism of the resource manager and how to deal with backlogs of queued-up data due to previous mismatches between demand and capacity. Further work addressed some of the limitations of the algorithm by better defining the actuation mechanism [Gho03, Gho04] and the application model [Kan07].

Buttazzo et al. [But98] introduced the elastic model where each task's rate can change within a certain interval. The quality of service delivered by each task is modeled as a spring with a given elastic coefficient. Tasks with a lower elastic coefficients will allow for larger variation in rate. The rates change when a task is added to or removed from the system. Further work deals with unknown and variable execution times [But02], optimal control, when the applications are controllers [But04], when dealing with energy minimization [Mar07], and for managing the utilization in computer networks [Ped03]. The problem of overloads was specifically addressed in [But04, But07]. The mechanism for solving overloads is based on acting as soon as an overload has been detected and on delaying the next job release of the overloading task until all pending jobs of this task get executed. The main criticism of this method is that the performance metric is hardwired into the algorithm and may not be changed. This has been addressed in [Hu06, Cha09] where the authors extended the elastic method to general metrics.

Another widely used category of methods for adaptation are server based methods first formalized in [Raj98]. In this formalism tasks are attached to servers which execute on the resource. Each server obtains a portion of the resource's capacity, thus acting as a virtual resource for the tasks assigned to it. Adaptation is done using a combination of two types of methods:

1. *resource reclaiming* which detects the unused capacity of idling servers and gives it to overloaded servers, and
2. *resource reservation* which decides on the reserved capacity allocated to each server.

This category of methods involves two levels of scheduling:

1. scheduling of the servers on the resource, and
2. scheduling of the tasks inside each server.

The adaptation mechanisms affect the first level only. Both these adaptation methods are part of the class of schedule adaptation mechanisms (class 4 in Section 2.2). Job dropping is typically required for each server in order to deal with overloads in the system. Earlier notions included the *constant utilization server* [Den97] and the *total bandwidth server* [Spu94, Spu96]. However, the most widely used formalism has been the *constant bandwidth server* (CBS) first described in [Abe98]. Various adaptation methods based on CBS have been proposed [Ce03a, Cer05, Liu07, Fon10, Fon11, Kha11, Kha13].

In [Ce03a, Cer05] the authors develop the control server model for scheduling and propose an approach to schedule control tasks in order to minimize jitter and latency. These methods tune the scheduler parameters such that jobs are schedulable as long as the incoming load in the system is below a certain bound (less or equal to 1) and they aim at gracefully degrading the quality-of-service experienced by the user when the incoming load is above the bound.

Liu et al. [Liu07, Yao08] presented a Recursive Least Squares based controller to control the utilization of a distributed system by means of rate adjustment. The authors consider distributed systems where tasks are schedulable if the utilization on each resource is kept at or below certain bounds. In [Liu07] the load on one resource is influenced by the load on the other resources via some coefficients which are estimated on-line, while in [Yao08] the model of the system is learned on-line.

In [Fon10, Fon11] the authors present a number of feedback-based algorithms for adaptive reservation. The aim is to control the delay properties of control tasks by measuring and reacting to the delay error (the difference between the prescribed and the measured delay). The actuation mechanism is adjustments in the parameters of the servers on which the tasks are running. The resource management policies are based on control theory and formal proofs of their stability are given.

Khalilzad et al. [Kha11, Kha13] presented a feedback-based control algorithm for scheduling multimedia tasks with extremely large variations in resource demand. The method adapts the parameters of a hierarchical schedule [Nol09] subject to variations in resource demand. A protection mechanism is devised based on the elastic model, to gracefully degrade the performance of the system in overload conditions.

Many resource management policies use ideas from control theory to control the resource demand and capacity in real time systems. Lu et al. [Lu02] described a framework for feedback control scheduling, where the source of non-determinism is execution time variation, and the actuation method is admission control and task rate change. The authors treat overloads by actuating based on utilization and deadline miss ratio. Both measures, however, saturate at 100% and thus are limited in their capability of describing overloads in the system.

Cervin et al. [Cer02] proposed to overcome the overload issue by using a feedback-feedforward resource manager, where small variations in execution times are handled by a feedback mechanism and large variations are handled, before they happen, by a feedforward mechanism. This means that this method is only applicable to systems where the application can warn the resource manager about large increases of execution times in advance.

Combaz et al. [Co05a, Co05b, Com08] proposed a QoS scheme for applications composed of tasks, each described as graphs of subtasks, and each subtask has a number of modes.

In [Abd03] the authors propose a model where the state of the system is composed of the sizes of queues where jobs accumulate before being executed and the goal of the adaptation mechanism is to keep these queues at a certain level of occupancy. This model is stemmed from the functioning of web servers. It is well suited for accurately describing situations where overloads occur. However, queue sizes are values that saturate at 0 (they are positive values) and the proposed model linearizes the behavior of queues to the region where they are not empty. This means that the resource manager must always keep the queues sizes at positive (not necessary small) levels. Since non-empty queue sizes are generally associated with overloaded systems, this means that the system is always kept at a certain level of overload. This behavior may not be acceptable for systems where it is important that end-to-end delays are kept small.

Palopoli et al. [Pa09a, Cuc10] consider resource-reservation schedulers and propose a feedback based technique for adjusting the quotas of tasks in reaction to task execution time variations. In [Cuc10] tasks share several resources and the quotas of tasks on all resources are determined together, in order to minimize end-to-end delays.

In all works regarding adaptive systems, the issue of stability arises. Stability has slightly different meaning for the different works, but it always includes the idea that a system is stable when it is able to avoid situations where jobs released for execution can accumulate in an unbounded way, without being executed in a timely fashion. The above related works deal with stability in several ways: by proposing methods derived from control theory [Lu05, Ce03a, Liu07, Yao08, Abd03, Pa09a, Cuc10, Fon11, Kha11], by developing custom analysis to the proposed methods [Lee98, But98, But02, But04, Mar07], and by empirical simulation [Lu02, Cer02]. All these works give specific solutions (methods for adaptivity together with their stability analysis) to specific types of problems. However, none of the above works provides hints about how to answer basic stability questions for generic adaptive distributed real-time system.

In this thesis we do not give a specific method for adaptation as in the related works. Instead we build a framework for modeling and analyzing generic real-time systems, with the goal of assuring worst-case stability of the system. We focus on making our framework as generic as possible in order for it to be appropriate for a large class of real-time systems. The stability analysis presented in this paper is done for adaptation mechanisms in the first class (*Job Flow Adaptation*), but we later show how it can be extended for methods from classes 2 and 3.

We do not address adaptation mechanisms from the last class (scheduler adaptation) as they have limited capacity for handling overload situations. If the system finds itself in a situation where the incoming load is above 1 for extended periods of time, methods from this class are powerless at preventing accumulations from happening and possibly leading to system crashes. Methods such as [Ce03a, Pa09a, Liu07, Com08, Yao08, Cuc10, Fon11, Kha11] need to relay on adaptation mechanisms from the other classes to handle such cases. We view these methods as being complementary with the ones from the former three classes [Lee98, But98, But02, But04, Mar07, Lu02, Abd03, Cer02], their goal being to improve system performance rather than assuring worst-case stability. This view is in agreement with the one presented in [Liu07].

2.4 Link with Queueing Networks

The theory presented in this paper has similarities with the theory on queueing networks [Ke79]. Queueing networks describe systems where *workers* service a queue of *packets* by using a work-*station*. For some systems, several workers must share a station. Such systems are called *multi-class queueing networks*. The concepts of packets, workers, and stations are depicted in Figure 2.2 where we have a queueing network composed of four workers chained together, where, workers 1 and 4 share station 1, and workers 2 and 3 share station 2.

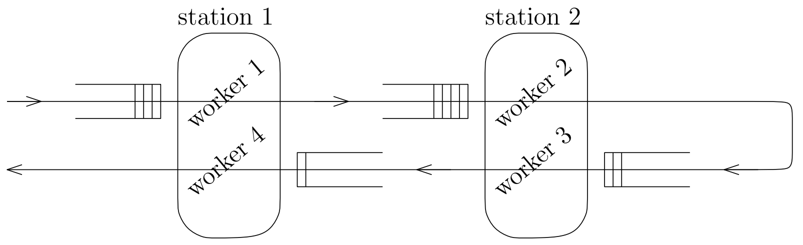


Figure 2.2: Example of a queueing network with two stations and four workers.

Packets arrive in the networks following a given distribution, with an average inter-arrival time of m (average arrival rate of $1/m$). Each worker services a package in an amount of time that varies stochastically according to a distribution, with a known average execution time. Queueing networks theory is used to model computer networks, telecommunication systems, distribution chains and warehouses, the manufacturing of products in factories, etc. (see [Bra08]).

The real-time systems that are described in this thesis can be modeled as queueing networks where workers are tasks, jobs are packets, and resources are stations. In Figure 2.3 we present the real-time system that is equivalent with the queueing network presented in Figure 2.2. In a real-time system, a job does not move from the queue of a task to the next, instead jobs of all tasks in that task graph are released for execution simultaneously[2], however, they become ready for execution only when their dependencies are satisfied. Dependencies are modeled by the links in the task graph, thus, for the example, in Figure 2.3 only the kth job of task τ_1 is ready for execution upon release. After its completion the kth job of τ_2 will become ready for execution, and so on. This behavior of jobs, however, simulates the behavior of packets (where packets move from worker to worker), thus allowing us to model real-time systems as queueing networks.

[2]A release models the event in a real-time system when the computation modeled by the software task graph is called for execution.

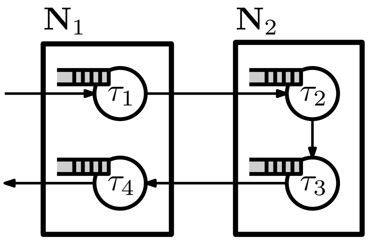


Figure 2.3: Example of a real-time system with two resources and four tasks. This example can be modeled as the queueing network in Figure 2.2.

We, therefore, express ourselves by saying that jobs move from one task to the next.

The main problem with queueing networks is determining if they are stable, that is, if packets exit the network at the same rate (on average) at which they enter. This is an important problem since queues are assumed to be of finite size and, if packets exit the network at a slower rate compared with their input rate, then the queues inside the network will overflow. This problem is formulated in two settings [Bra08]:

- *stochastic stability of a queueing network*, where both input rates and execution times vary, according with certain stochastic distributions, and

- *deterministic stability of a queueing network*, where the input rates and execution times are assumed fixed at given values.

There exist several solutions to these problems, in both settings, for simple instances of queueing networks which can be modeled as markov processes [Ke79] or continuous flow models [Dai96]. For more advanced networks, such as multi-class queueing networks, however, there are results only for particular example systems [Kum95]. An important result in queueing networks theory is the *subcriticality condition* for a queueing network, which is a necessary condition for stability, generally applicable to any network [Bra08]. A series of papers in the early 1990s [Kum90, Lu91, Ryb92] have shown that there exists a need for general solutions to the above problems as the number of subcritical unstable queueing networks is potentially very large.

In the theory that we develop in this thesis, the modeling that we propose has some similarities with modeling applied to queueing networks, and the necessary condition for stability that we derive in Section 5.1 has a similar meaning with the notion of subcriticality of queueing networks. However, in this thesis we deal with absolute stability, that is, we bound the behavior of the system in the worst case, while in queueing networks theory stochastic stability is used, where only bounds on the expected behavior are determined. The results that we obtain in Section 5.2 provide a general solution to the deterministic stability problem described above. Furthermore, the rest of the results presented in this thesis (Section 5.3) use the concept of resource manager and have no counterpart in results known from queueing networks theory.

3 Preliminaries

In this chapter we introduce the necessary notations used throughout the thesis to describe and model the system. We also introduce the control theoretic notions of stability that we later use in our stability analysis.

3.1 Mathematical Notations

In this thesis we use standard mathematical notations. We describe a set of elements as: $\mathfrak{S} = \{s_i, i \in \mathcal{I}_{\mathfrak{S}}\}$ where $\mathcal{I}_{\mathfrak{S}} \subset \mathbb{Z}_+$ is the index set of $\mathfrak{S}$. We make the convention that if $\mathfrak{S} = \{s_i, i \in \mathcal{I}_{\mathfrak{S}}\}$ is an ordered set, then s_i appears before $s_{i'}$ in the set if and only if $i < i'$, for all $s_i, s_{i'} \in \mathfrak{S}$, $s_i \neq s_{i'}$. $\mathcal{P}(\mathfrak{S})$ is the power set of $\mathfrak{S}$.

We denote a $n \times m$ matrix as $A = [a_{i,j}]_{n\times m}$ where n is the number of rows and m is the number of columns. We denote with $0_{n\times m}$ the matrix whose elements are all 0 and with I_n the $n \times n$ identity matrix. We denote a column vector as $\vec{v} = (v_1, v_2, \cdots v_n)^T$ or more compactly as $\vec{v} = [v_i]_n$. We denote with 0_n and 1_n the n-dimensional column vectors formed of all elements equal to 0 and 1 respectively. In an n-dimensional normed space $\mathcal{V}$ we denote the norm of a vector $\vec{v}$ with $|\vec{v}|$. For a given matrix A, its *kernel space* is $\mathrm{Ker}(A) = \{x|Ax = 0\}$ and its *image space* is $\mathrm{Im}(a) = \{x|Ax = x\}$. We say that a matrix A is *idempotent* if $A^2 = A$.

When we compare two vectors $\vec{v} = [v_i]_n$ and $\vec{u} = [u_i]_n$, we use the notations $\succ$, $\prec$, $\succeq$, and $\preceq$. For example the relationship $\vec{v} \succeq \vec{u}$ means $v_i \geq u_i$, $\forall i \in \{1, 2, \cdots, n\}$. Similarly for the rest of the notations as well.

We recall that a function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a $\mathfrak{K}$-*function* if it is continuous, strictly increasing, and $\gamma(0) = 0$. A function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a $\mathfrak{K}_\infty$-*function* if it is a $\mathfrak{K}$-function and it is unbounded. A function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a $\mathfrak{KL}$-*function* if for each fixed $t \geq 0$ the function $\beta(\cdot, t)$ is a $\mathfrak{K}$-function and for each fixed $s \geq 0$ the function $\beta(s, \cdot)$ is decreasing and $\beta(s, t) \to 0$ as $t \to \infty$.

3.2 Description of the System

We consider that our distributed platform is composed of a finite set of n resources (e.g. processors and buses) $\mathfrak{R} = \{\mathbf{N}_i, i \in \mathcal{I}_{\mathfrak{R}}\}$, $\mathcal{I}_{\mathfrak{R}} = \{1, \cdots, n\}$. Each resource is characterized by the finite set of tasks that compete for it $\mathbf{N}_i = \{\tau_j, j \in \mathcal{I}_{\mathbf{N}_i}\}$. In this thesis we only consider time-shared mutual-exclusive resources, where only one task may use the resource at any given time. Each resource is, therefore, equipped with a scheduler to schedule the succession of tasks. The scheduler is embedded (as part of the hardware or the software middleware) in the resource.

The applications running on this platform are a finite set of acyclic task chains $\mathfrak{A} = \{\mathfrak{C}_p, p \in \mathcal{I}_{\mathfrak{A}}\}$, formed of tasks linked together through data dependencies[1]. A task chain is a finite ordered set of tasks: $\mathfrak{C}_p =$

[1]In Section 6.2 we will generalize application models to acyclic task graphs where tasks may have multiple data dependencies to their predecessors, and their output may be a dependency for several successor tasks.

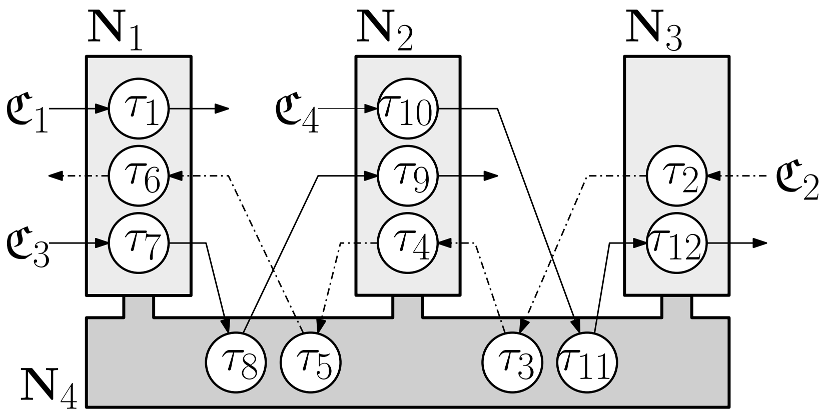


Figure 3.1: Distributed system example.

$\{\tau_j, j \in \mathcal{I}_{\mathfrak{C}_p}\}$. With each task chain $\mathfrak{C}_p$ we associate the set of rates at which it can release new instances for execution: $\mathbf{P}_p \subseteq [\rho_p^{\min}, \rho_p^{\max}]$. The tasks of all task chains are mapped on the resources of the system.

Each task in the task chain $\mathfrak{C}_p$ has at most one data dependency to the previous task in $\mathfrak{C}_p$. Figure 3.1 shows an example of a system with 4 resources $\mathfrak{R} = \{\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \mathbf{N}_4\}$, where $\mathbf{N}_1 = \{\tau_1, \tau_6, \tau_7\}, \mathbf{N}_2 = \{\tau_4, \tau_9, \tau_{10}\}$, and $\mathbf{N}_3 = \{\tau_2, \tau_{12}\}$ are processors and $\mathbf{N}_4 = \{\tau_3, \tau_5, \tau_8, \tau_{11}\}$ is a communication link. The application is composed of 4 task chains $\mathfrak{A} = \{\mathfrak{C}_1, \mathfrak{C}_2, \mathfrak{C}_3, \mathfrak{C}_4\}$, where $\mathfrak{C}_1 = \{\tau_1\}$ contains only one task, $\mathfrak{C}_2 = \{\tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$ contains 5 tasks and is spread on all resources, and $\mathfrak{C}_3 = \{\tau_7, \tau_8, \tau_9\}$ and $\mathfrak{C}_4 = \{\tau_{10}, \tau_{11}, \tau_{12}\}$ contain 3 tasks and are spread on $\{\mathbf{N}_1, \mathbf{N}_4, \mathbf{N}_2\}$ and $\{\mathbf{N}_2, \mathbf{N}_4, \mathbf{N}_3\}$ respectively. Observe that we treat both processors and communication links as resources. Because of this, messages sent on communication links are called tasks in our notation. In Figure 3.1, τ_3, τ_5, τ_8, and τ_{11} are, in fact, messages since $\mathbf{N}_4$ is a communication link.

All tasks of a task chain $\mathfrak{C}_p$ release jobs periodically and simultaneously[2], at the variable rate ρ_p chosen from the set $\mathbf{P}_p$. We de-

[2]To describe this phenomenon, we also say that an instance of the task chain $\mathfrak{C}_p$ is released at a rate ρ_p.

note the kth job of task τ_j as τ_{jk}. However, jobs cannot be executed before their data dependency is solved. We define a task τ_j by the interval of execution times that jobs of this task can have $\tau_j = (\mathbf{C}_j = [c_j^{\min}, c_j^{\max}])$. A task may only belong to one task chain and may only consume one resource. The set of all tasks in the application is $\mathbf{\Theta} = \bigcup_{p \in \mathcal{I}_{\mathfrak{A}}} \mathfrak{C}_p$. We denote the index set of all tasks in the application as $\mathcal{I}_{\mathbf{\Theta}}$. Also, we denote with $\mathbf{P} = \prod_{p \in \mathcal{I}_{\mathfrak{A}}} \mathbf{P}_p$ the space of rates and with $\vec{\rho}$ a vector in this space.

Based on the above defined sets and assuming the existence of element $\xi \notin \mathcal{I}_{\mathbf{\Theta}}$ we define the following mappings:

- $\gamma : \mathcal{I}_{\mathbf{\Theta}} \rightarrow \mathcal{I}_{\mathfrak{A}}$, $\gamma(j) = p$ if $\tau_j \in \mathfrak{C}_p$, which is a mapping from tasks to task chains,

  Example: $\gamma(5) = 2$ in the example from Figure 3.1 since τ_5 belong to $\mathfrak{C}_2$. We then use $\mathfrak{C}_{\gamma(5)}$ when dealing with $\mathfrak{C}_2$.□

- $\nu : \mathcal{I}_{\mathbf{\Theta}} \rightarrow \mathcal{I}_{\mathfrak{R}}$, $\nu(j) = i$ if $\tau_j \in \mathbf{N}_i$ which is a mapping from tasks to resources,

  Example: $\nu(5) = 4$ in the example from Figure 3.1 since τ_5 is mapped to resource $\mathbf{N}_4$. We then use $\mathbf{N}_{\nu(5)}$ when dealing with $\mathbf{N}_4$. □

- $\pi : \mathcal{I}_{\mathbf{\Theta}} \rightarrow \mathcal{P}(\mathcal{I}_{\mathbf{\Theta}})$, $\pi(j) = \mathcal{S} \subset \mathcal{I}_{\mathfrak{C}_p}$ is the set of all indexes of tasks that are predecessors (direct or indirect) of τ_j in the task chain,

  Example: $\pi(5) = \{2, 3, 4\}$ in Figure 3.1 since tasks τ_2, τ_3, and τ_4 are the predecessors of task τ_5 in $\mathfrak{C}_2$. □

- $\varphi : \mathcal{I}_{\mathfrak{R}} \rightarrow \mathcal{P}(\mathcal{I}_{\mathfrak{R}})$, $\varphi(i) = \mathcal{S}_{\mathcal{N}} \subset \mathcal{I}_{\mathfrak{R}}$, is the set of all indexes of resources whose tasks have successors on resource $\mathbf{N}_i$,

  Example: $\varphi(4) = \{1, 2, 3, 4\}$ in Figure 3.1 because tasks running on $\mathbf{N}_4$ have predecessors from all resources in the system:

- τ_7 on $\mathbf{N}_1$ is a predecessor of τ_8 on $\mathbf{N}_4$,
- τ_4 and τ_{10} on $\mathbf{N}_2$ are the predecessor of τ_5 and τ_{11} respectively, on $\mathbf{N}_4$,
- τ_2 on $\mathbf{N}_3$ is a predecessor of τ_3 on $\mathbf{N}_4$, and
- τ_3 on $\mathbf{N}_4$ is a predecessor of τ_5 on $\mathbf{N}_4$. □

- $\mathfrak{p} : \mathcal{I}_{\mathbf{\Theta}} \to \mathcal{I}_{\mathbf{\Theta}} \cup \{\xi\}$, $\mathfrak{p}(j) = j' \neq \xi$ if $\tau_{j'}$ is the predecessor of τ_j, or $\mathfrak{p}(j) = \xi$ if τ_j has no predecessor.

  Example: In Figure 3.1, $\mathfrak{p}(5) = 4$ since τ_4 is the direct predecessor of τ_5 in $\mathfrak{C}_2$, while $\mathfrak{p}(10) = \xi$ since τ_{10} does not have a predecessor in $\mathfrak{C}_4$. □

Using the above mappings, the predecessor task of τ_j (assuming τ_j has one) is $\tau_{\mathfrak{p}(j)}$ and the release rate of the task chain, of which τ_j is part of, is $\rho_{\gamma(j)}$. The resource on which the predecessor of τ_j is running is $\mathbf{N}_{\nu(\mathfrak{p}(j))}$.

A job of a task in the system is the tuple: $\tau_{jk} = (c_{jk}, \rho_{jk}, r_{jk})$ where: c_{jk}, ρ_{jk}, and r_{jk}, are the execution time, rate, and response time of the kth job of task τ_j. The response time of any job of a task represents the interval of time between the release and the finish time of the job. A job τ_{jk} can be in one of the following states:

1. *Released* when it has been released at the rate of the task chain,
2. *Ready for Execution* when the job's data dependency was solved, that is, when the corresponding job of the predecessor of this task $\tau_{\mathfrak{p}(j)k}$ has finished executing,
3. *Under Execution* when the job has been partially executed, that is, when it occupied the resource for a portion of its execution time, and
4. *Finished* when the job has been fully executed.

The tasks in the system are scheduled, on their particular resources, using any scheduler which satisfies the following properties:

1. it is *non-idling*: it does not leave the resource idle if there are pending jobs;
2. it executes successive jobs of the same task in the order of their release.

At a certain moment in time, due to the functioning of the schedulers, at most one job of any task may be under execution. We consider that all jobs which are ready for execution and under execution are accumulated in queues, one for each task in the application. For tasks that have no data dependencies, a job becomes ready for execution whenever it is released. Whenever a job finishes its execution, it is removed from its task's queue. At the same time, the corresponding job of the task's successor becomes ready for execution and, thus, gets added to the successor task's queue. We assume that this event takes place instantaneously. This is acceptable because we treat communication links as resources, which means that data dependencies are just virtual constructs in the model.

3.3 Resource Manager

The system features a resource manager whose goal, among others, is to measure execution times and, then, adjust task chain release rates such that the jobs pending on all resources are executed in a timely fashion and the amount of time spent in overload situations is minimized. We consider the system stable if, under the worst possible run-time scenario, the overload in the system is kept finite, meaning that the system does not keep accumulating jobs without having a way of executing them. The resource manager is part of the middleware of the system and is, in general, distributed over all resources.

Whenever the resource manager is activated, we assume that it has a worst-case response time of $\Delta < h$, where h is its actuation period, and that it has a worst-case execution time on each resource of less than Δ. We assume that the resource manager imposes the newly computed task chain rates simultaneously, on all resources, at Δ after it has been activated. This means that all jobs released *during* the running of the resource manager are still released at the old task chain rates. We treat all parts of the resource manager as tasks from the application set, and we include them in the task sets $\mathbf{N}_i$ on their particular resources.

The resource manager, in general, will be tailored to the application at hand and will function such as to optimize specific quality metrics related to the goal of the particular system, as discussed in Chapter 2. However, from the point of view of this theoretical framework, we do not impose any constraints on the structure and function of the resource manager, apart from actuating periodically and having the goal of keeping the task queues bounded by means of task chain rate adjustment. In chapter 6 we shall extend the resource manager definition by allowing for more flexible actuation methods.

3.4 Stability of Discrete-Time Dynamical Systems

A discrete-time control system is part of a larger class of systems called discrete-time dynamical systems [Mic08]. A discrete-time dynamical system is a tuple $\{\mathcal{T}, \mathcal{X}, \mathcal{A}, \mathcal{S}, \mathcal{U}\}$ where: $\mathcal{T} = \{t_{[k]} | k \in \mathbb{Z}_+, 0 = t_{[0]} < t_{[1]} < \cdots < t_{[k]} < \cdots\}$ is the discrete set of times at which the system evolves, $\mathcal{X}$ is the state space, $\mathcal{A} \subset \mathcal{X}$ is the set of all initial states of the system ($\vec{x}_{[0]}$), $\mathcal{U}$ is the bounded set of all inputs and $\mathcal{S}$ is the set of all trajectories (all solutions of (3.1) : $\vec{x}_{[k]} = p(k, \vec{x}_{[0]}, \vec{u}_{[k]})$ where $t_{[k]} \in \mathcal{T}$, $\vec{x}_{[0]} \in \mathcal{A}$, and $\vec{u}_{[k]} \in \mathcal{U}$). The state space must be a normed space $(\mathcal{X}, |\cdot|)$. A dynamical system's state is desired to be 0_n. Because systems are subject to inputs, this condition does not hold. Under this premise a dynamical system is said to be stable if its state remains "close" to 0_n for all input patterns and initial conditions.

For our system we consider the following stability definition (we recall that notations, such as $\mathfrak{K}$ and $\mathfrak{KL}$-functions are described in Section 3.1):

Definition 1 (Global Asymptotic Stability):
A dynamical system S, expressed recursively as:

$$F(\vec{x}_{[k+1]}, \vec{x}_{[k]}, \vec{u}_{[k]}) = 0_n \tag{3.1}$$

is *global asymptotically stable* (GAS) [Son01] if there exists a $\mathfrak{KL}$-function β such that for each initial state $\vec{x}_{[0]} \in \mathcal{A}$ and for each input function $\vec{u} : \mathbb{Z}_+ \rightarrow \mathcal{U}$ we have that:

$$|p(k, \vec{x}_{[0]}, \vec{u}_{[k]})| \leq \beta(|\vec{x}_{[0]}|, k) \tag{3.2}$$

$\diamond$

Definition 2 (Input-to-State Stability):

A dynamical system S (equation (3.1)) is *input-to-state stable* (ISS) [Jia01] if there exists a $\mathfrak{KL}$-function β and a $\mathfrak{K}$-function γ such that for each initial state $\vec{x}_{[0]} \in \mathcal{A}$ and for each input function $\vec{u} : \mathbb{Z}_+ \rightarrow \mathcal{U}$: we have that:

$$|p(k, \vec{x}_{[0]}, \vec{u}_{[k]})| \leq \max\{\beta(|\vec{x}_{[0]}|, k), \gamma(||\vec{u}||)\} \tag{3.3}$$

where $||\vec{u}|| = \sup\{|\vec{u}_{[k]}|, k \in \mathbb{Z}_+\}$. $\diamond$

A GAS system approaches its equilibrium point regardless of the initial state from where it begins. An ISS system initially approaches its equilibrium point similarly to a GAS system but it stops when its state becomes bounded in a ball of a certain size around its equilibrium point. The size of the ball depends on the magnitude of its input. We illustrate graphically the meaning of these stability concepts in Figures 3.2a for GAS and 3.2b for ISS. For a deeper understanding of these two concepts and the link between them we point the reader

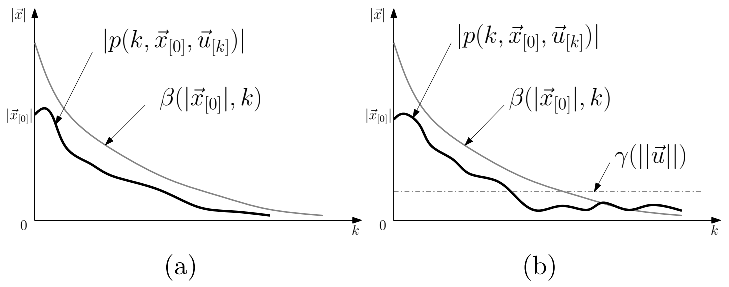


Figure 3.2: (a) GAS: The system's state approaches 0_n as time passes, regardless of the initial state. (b)ISS: The system's state reduces at first, but then becomes trapped below a bound determined by the magnitude of the inputs.

to works such as [Son01, Jia01]. We will only note here that a ISS system is GAS if its input is null and that the ISS property (3.3) may be written in other ways (of course, for different β and γ functions), e.g.:

$$|p(k, \vec{x}_{[0]}, \vec{u}_{[k]})| \leq \beta(|\vec{x}_{[0]}|, k) + \gamma(||\vec{u}||)$$

For the system presented in this thesis (modeled as (4.1)) we derive conditions under which it is GAS with respect to its controlled inputs ($\vec{u}$) and ISS with respect to its perturbations ($\vec{w}$). The norm of the state of our system will finally become bounded in a ball around 0_n. In our case, this means that task queues will be bounded. This implicitly guarantees various real-time properties e.g. bounded response times and end-to-end delays. The system, if stable, will behave according to the following stability definition, taken from [Mic08]:

Definition 3 (Ultimate Boundedness):

A dynamical system S is ultimately bounded if there exists $\Psi > 0$ and if corresponding to any $\alpha > 0$ and $t_0 \in \mathcal{T}$, there exists $t' \in \mathcal{T}, (t'(\alpha))$ independent of t_0 such that for all $p(t_{[k]}, \vec{x}_{[0]}) \in \mathcal{S}$ we have that $|p(t_{[k]})| \leq \Psi$, $\forall t \geq t'$ whenever $|\vec{x}_{[0]}| \leq \alpha$. $\diamond$

Any system that satisfies the above definition is stable, and this means that its state becomes trapped in the ball of size Ψ around 0_n, after a certain amount of time, regardless of the initial state $\vec{x}_{[0]}$. Each parameter of the state, therefore, will become constrained in a bounded interval. In our case, this means that task queues will be bounded.

4

Modeling of the Adaptive Distributed Real-Time System

IN THIS chapter we develop a model capturing the evolution in time of the adaptive distributed real-time system described in the previous chapter. We continue by determining and modeling its worst-case behavior which shall be used for developing our main results in Chapter 5. We end the chapter with an illustrative example and an informal description meant to help with better understanding of the system model.

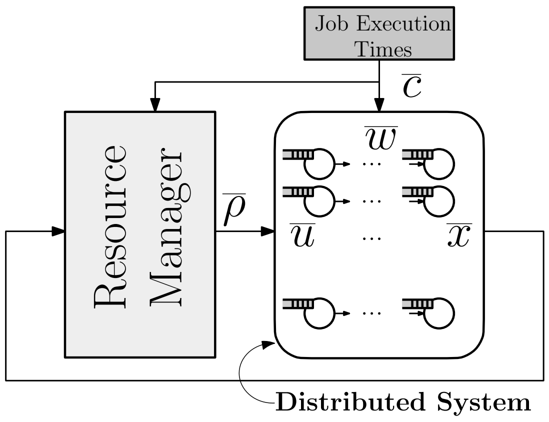


Figure 4.1: Control theoretic view of our adaptive distributed system.

4.1 Control Theoretic View of a Real-Time System and its Parameters

We model our distributed real-time system as a discrete-time control system, described by a system of difference equations:

$$F(\vec{x}_{[k+1]}, \vec{x}_{[k]}, \vec{u}_{[k]}, \vec{w}_{[k]}) = 0_n \tag{4.1}$$

Where $\vec{x}_{[k+1]}$ and $\vec{x}_{[k]}$ are the state vectors of the system at the future ($t_{[k+1]}$) and the current ($t_{[k]}$) time moments, $\vec{u}_{[k]}$ is the current input, and $\vec{w}_{[k]}$, is the current perturbation experienced by the system. In our system, input is provided by the resource manager as the task chain rates vector and the perturbations come from the variation in execution times of jobs. Finally, the state of the system is composed of all other time varying quantities that appear in equation (4.1). This will boil down to the state being represented by the vector of queue sizes of each task in the system.

The model of our system can be depicted as in Figure 4.1. While the tasks and the resource manager are distributed over the resources of the system, from a modeling perspective the tasks and resources

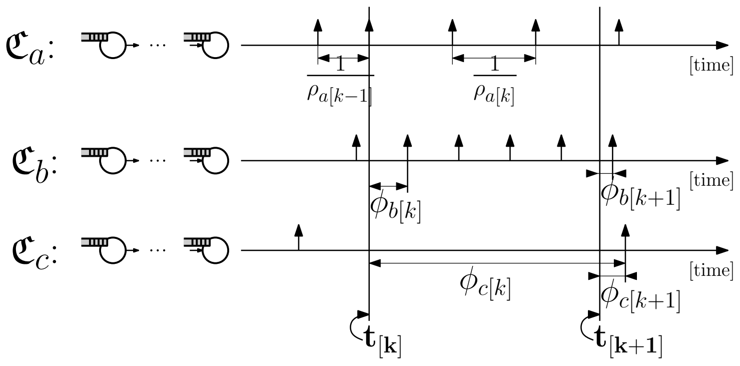


Figure 4.2: Examples of task chain release patterns.

form the plant, and the resource manager is the controller controlling the accumulation of execution times by means of adjusting task chain rates.

We describe the evolution of our system at discrete moments in time $t_{[k]}$, $k \in \mathbb{Z}_+$ when one of the following events happens:

1. the resource manager activates,
2. a job of a task τ_i is released at a different rate than the former job of this task, or,
3. the scheduler on any resource switches to executing jobs of a different task than before.

To help with understanding, we show the meaning of the quantities ρ_p, $\forall p \in \mathcal{I}_{\mathfrak{A}}$, representing the rates of the task chains in the system, with the help of Figure 4.2, where we show the evolution of an example system with three task chains $\mathfrak{C}_a$, $\mathfrak{C}_b$, and $\mathfrak{C}_c$ between time moments $t_{[k]}$ and $t_{[k+1]}$. The meaning of the task chain rates is simply that the distance between successive job releases of tasks of a task chain $\mathfrak{C}_p$ is $1/\rho_p$. We can see from $\mathfrak{C}_a$ that jobs released after $t_{[k]}$ are separated by $1/\rho_{a[k]}$ while jobs released before are separated by $1/\rho_{a[k-1]}$. In the case of $\mathfrak{C}_b$ we can observe that $t_{[k]}$ does not correspond with the release of new jobs, thus, there exists an offset $\phi_{b[k]}$

between $t_{[k]}$ and the first jobs released after it. From $\mathfrak{C}_c$ we observe that we might not have any jobs released during $[t_{[k]}, t_{[k+1]}]$.

4.2 System Model

In this section we develop the system of difference equations that form the model of our distributed system.

The overload situation in a system is characterized by the queues in which jobs accumulate before they get executed. The evolution of the queues of tasks running on a given resource $\mathbf{N}_i$, at the discrete moments in time defined in the previous section, depends on:

1. the number of jobs arriving to these queues,
2. the execution times of all these jobs, and
3. how the scheduler on that resource decides to execute jobs from these queues.

We model the evolution of queues of all tasks running on a resource as an accumulation of execution time that needs to be executed by the resource. For each resource $\mathbf{N}_i$, $i \in \mathcal{I}_{\mathfrak{R}}$ we have:

$$\begin{aligned}\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]} = \\ \max\left\{ 0, \quad \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k]} + \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} s_{j[k+1]} - \right. \\ \left. (t_{[k+1]} - t_{[k]}) \right\}\end{aligned} \tag{4.2}$$

The quantities appearing in the above equation have the following meaning:

- $q_{j[k]}$, $\forall j \in \mathcal{I}_{\mathbf{\Theta}}$ represent the queue sizes of the queues of all tasks in the system,

- $\sum\limits_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k]}$ is the accumulation of execution time existing in the queues of the tasks running on resource $\mathbf{N}_i$, at $t_{[k]}$

- $\sum\limits_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} s_{j[k+1]}$ is the amount of execution time of all incoming jobs to these queues, and

- $\sum\limits_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]}$ is the amount of execution time remaining in the queues at $t_{[k+1]}$.

In between the two consecutive moments of time $t_{[k]}$ and $t_{[k+1]}$, the resource can execute at most an amount of execution time of size $t_{[k+1]} - t_{[k]}$. Observe that we have modeled queue sizes as continuous values. This is in order to accurately represent situations where jobs remaining in queues are partially executed[1].

In equation (4.2) the quantities $s_{j[k+1]}$, $\forall j \in \mathcal{I}_{\mathbf{\Theta}}$ represent the number of jobs of τ_j that become ready for execution during $[t_{[k]}, t_{[k+1]}]$ and their evolution is described in the following equation:

$$s_{j[k+1]} = \begin{cases} \lceil q_{j'[k]} \rceil + s_{j'[k+1]} - \lceil q_{j'[k+1]} \rceil, & \text{if } j' = \mathfrak{p}(j) \neq \xi \\ \lceil \rho_{\gamma(j)[k]} \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{\gamma(j)[k]}\} \rceil, & \text{otherwise} \end{cases} \tag{4.3}$$

If τ_j has $\tau_{j'}$ as predecessor then $s_{j[k+1]}$ represents the number of jobs of $\tau_{j'}$ that were executed during $[t_{[k]}, t_{[k+1]}]$ (first branch of the equation). If τ_j has no predecessor, then $s_{j[k+1]}$ represents the number of jobs released by the task chain (second branch of the equation).

The quantities $\phi_{p[k]}$, $\forall p \in \mathcal{I}_{\mathfrak{A}}$ represent the offsets from $t_{[k]}$ when the new instance of task chains $\mathfrak{C}_p$ gets released into the system (see

[1]Modeling queues in this fashion is useful for our purpose of analyzing systems for stability, however, it does not impose any constraints on the functioning of systems. The resource manager of a system does not need to measure and utilize continuous queue size values in order for this modeling and analysis to be applicable.

Figure 4.2) and their evolution in time is given by equation:

$$\phi_{p[k+1]} = \phi_{p[k]} + \frac{1}{\rho_{p[k]}} \lceil \rho_{p[k]} \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{p[k]}\} \rceil - (t_{[k+1]} - t_{[k]}), \tag{4.4}$$

The quantities $c_{j[k]}$, $\forall j \in \mathcal{I}_{\boldsymbol{\Theta}}$ represent the average execution times of tasks τ_j during the time interval $[t_{[k]}, t_{[k+1]}]$. These quantities are unknown, therefore we will treat them as disturbances. The quantities $\rho_{p[k]}$, $\forall p \in \mathcal{I}_{\mathfrak{A}}$ represent the rates at which task chains release new instances for execution and are the inputs (decided by the resource manager) to our real-time system.

Equations (4.2) to (4.4) form a preliminary candidate for the model of our system. For ease of use, we reduce this system as described below. By observing from equation (4.4) that

$$\begin{aligned} \lceil \rho_{p[k]} \cdot \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{p[k]}\} \rceil =& \rho_{p[k]} \cdot (\phi_{p[k+1]} - \phi_{p[k]}) + \\ & \rho_{p[k]} \cdot (t_{[k+1]} - t_{[k]}) \end{aligned}$$

and by introducing the expressions of $s_{j[k]}$ into equation (4.2) we obtain:

$$\begin{aligned} &\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]} = \\ &\qquad \max\Bigg\{ 0, \quad \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k]} + \\ &\qquad\qquad \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \Bigg(\sum_{j' \in \pi(j)} \Big(\lceil q_{j'[k]} \rceil - \lceil q_{j'[k+1]} \rceil \Big) \Bigg) - \\ &\qquad\qquad \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \rho_{\gamma(j)[k]} \big(\phi_{\gamma(j)[k]} - \phi_{\gamma(j)[k+1]} \big) + \\ &\qquad\qquad \Bigg(\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \rho_{\gamma(j)[k]} - 1 \Bigg) \big(t_{[k+1]} - t_{[k]} \big) \quad \Bigg\} \end{aligned} \tag{4.5}$$

Equation (4.5), *repeated for every resource* $\mathbf{N}_i$, $i \in \mathcal{I}_{\mathfrak{R}}$, *forms the model of our system.*

The task chain release rates are the inputs to the system, given by the resource manager:

$$\vec{\rho}_{[k]} = f_c(\vec{x}_{[k]}) \tag{4.6}$$

where $f_c : \mathcal{X} \rightarrow \mathbf{P}$ is the model of the resource manager. In this work, we do not concern ourselves with the particular aspect of $f_c(\cdot)$ or its performance when applied to a specific system or class of systems. Instead, our goal is *to find conditions on the perturbations, inputs, and on the control law ($f_c(\cdot)$) which lead to a stable system.*

Stability implies that the state evolves in a bounded interval. Because we do not have enough equations in our model (we have one equation for each resource, not one equation for each task) we cannot determine the precise queue size for each queue, but we shall reason about the stability of the system nonetheless. We do so by aggregating the queues of all tasks running on a resource into the accumulation of execution times on that resource and bounded accumulations will imply bounded queues.

4.3 Worst-case Behavior of the System

We shall now perform a number of manipulations to the system model in order to make it easier to handle. First, we define a set of notations in order to make the formulas more compact to write. Next we develop an approximative version of the system that describes its behavior in the worst case. This allows us to eliminate the effect of variations in execution times. Finally, we write the whole system model in a matrix form.

In order to simplify the handling of equation (4.5) we define several notations, for all resources $\mathbf{N}_i$, $\forall i \in \mathcal{I}_{\mathfrak{R}}$. We denote with $\sigma_{i[k]}$ the accumulation of execution times on resource $\mathbf{N}_i$ at time moment $t_{[k]}$:

$$\sigma_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot q_{j[k]}$$

We denote with $\zeta_{i[k]}$ the accumulation of execution times flowing into $\mathbf{N}_i$ from tasks predecessor to the ones running on $\mathbf{N}_i$:

$$\zeta_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \left(\sum_{j' \in \pi(j)} \lceil q_{j'[k]} \rceil \right)$$

We use $o_{i[k]}$ to represents the accumulation of execution times determined by the offsets:

$$o_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \rho_{\gamma(j)[k-1]} \phi_{\gamma(j)[k]}$$

Finally, we use the value $U_{i[k]}$ to represents the load added to the system during the time interval $[t_{[k]}, t_{[k+1]}]$:

$$U_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \rho_{\gamma(j)[k]}$$

With the above notations, we can rewrite equation (4.5) as:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]} = \max \left\{ \begin{array}{ll} 0, & \sigma_{i[k]} + \zeta_{i[k]} - \zeta_{i[k+1]} + o_{i[k+1]} - o_{i[k]} + \\ & (U_{i[k]} - 1)(t_{[k+1]} - t_{[k]}) \end{array} \right\} \tag{4.7}$$

Because of the way we define the time points $t_{[k]}$ (see Section 4.1), we are sure that if $\phi_{j[k]} \neq 0$ then $\rho_{\gamma(j)[k-1]} = \rho_{\gamma(j)[k]}$ and thus

$$o_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \cdot \rho_{\gamma(j)[k]} \phi_{\gamma(j)[k]}$$

as well. All values $\sigma_{i[k]}$, $\zeta_{i[k]}$, $o_{i[k]}$, $U_{i[k]}$ are positive values for all $i \in \mathcal{I}_{\mathfrak{R}}$ and $t_{[k]}$, and are upper bounded by $\sigma^{\max}_{i[k]}$, $\zeta^{\max}_{i[k]}$, $o^{\max}_{i[k]}$, and $U^{\max}_{i[k]}$, respectively:

$$\sigma^{\max}_{i[k]} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_j \cdot q_{j[k]} \tag{4.8}$$

$$\zeta_{i[k]}^{\max} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \left(\sum_{j' \in \pi(j)} \lceil q_{j'[k]} \rceil \right) \tag{4.9}$$

$$o_{i[k]}^{\max} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(j)[k-1]} \phi_{\gamma(j)[k]} \tag{4.10}$$

$$U_{i[k]}^{\max} = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(j)[k]} \tag{4.11}$$

Let us observe that $o_{i[k]}^{\max} \leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max}$ for all $t_{[k]}$. We claim that the following inequality holds for every $\mathbf{N}_i$ at every $t_{[k]}$ and represents the worst-case behavior of the system:

$$\sigma_{i[k+1]}^{\max} \leq \max \left\{ \begin{array}{l} 0, \quad \sigma_{i[k]}^{\max} + \zeta_{i[k]}^{\max} - \zeta_{i[k+1]}^{\max} + o_{i[k+1]}^{\max} - o_{i[k]}^{\max} + \\ \qquad (U_{i[k]}^{\max} - 1)(t_{[k+1]} - t_{[k]}) \end{array} \right\} \tag{4.12}$$

The worst-case behavior happens when all jobs of all tasks have their worst-case execution times. We give the proof of inequality (4.12) in Section 4.3.2.

The state of the system in the original model (equation 4.5) is composed of the values of the queue sizes of all tasks in the system. For the worst-case behavior model derived above (equation 4.12) the corresponding state is $(\sigma_i^{\max} + \zeta_i^{\max})$ since these quantities contain the task queue sizes, whose evolution is of interest to us.

We observe that each resource $\mathbf{N}_i$ has two modes of operation, corresponding to the two arguments to the max operator in inequation (4.12):

1. *starving:* when all the queues of tasks running on the resource are empty, and the resource must sit idle ($\sigma_{i[k+1]}^{\max} = 0$), and

2. *non-starving:* when at least one of the queues of tasks running on the resource are non-empty and the resource is working executing jobs.

We rewrite inequation (4.12) (for each resource $\mathbf{N}_i$, $i \in \mathcal{I}_{\mathfrak{R}}$) in such a way as to highlight the state of the system and the two modes of operation:

$$\sigma^{\max}_{i[k+1]} + \zeta^{\max}_{i[k+1]} \leq \begin{cases} \zeta^{\max}_{i[k+1]}, & \text{if } \sigma^{\max}_{i[k+1]} = 0 \\ \sigma^{\max}_{i[k]} + \zeta^{\max}_{i[k]} + o^{\max}_{i[k+1]} - o^{\max}_{i[k]} + \\ \qquad (U^{\max}_{i[k]} - 1)(t_{[k+1]} - t_{[k]}), & \text{otherwise} \end{cases} \tag{4.13}$$

The evolution of the state of the system (associated with $\mathbf{N}_i$) is clear when the resource $\mathbf{N}_i$ is non-starving (the second branch), and it depends on its current state ($\sigma^{\max}_{i[k]} + \zeta^{\max}_{i[k]}$), the load added to $\mathbf{N}_i$ during the time interval $[t_{[k]}, t_{[k+1]}]$ ($U^{\max}_{i[k]}$) and a random but bounded noise due to the offsets ($o^{\max}_{i[k+1]} - o^{\max}_{i[k]}$). However when $\mathbf{N}_i$ is in starving mode, inequation (4.13) becomes indeterminate ($\sigma^{\max}_{i[k+1]} + \zeta^{\max}_{i[k+1]} \leq \zeta^{\max}_{i[k+1]}$ when $\sigma^{\max}_{i[k+1]} = 0$). We deal with this situation by finding an upper bound on $\zeta^{\max}_{i[k+1]}$ which depends on the other resources in the system (specifically, on the resources containing the predecessors of the tasks running on $\mathbf{N}_i$).

Considering the following definitions of the coefficients α and β for each resource $\mathbf{N}_i$ in the system:

$$\alpha_{ii'} = \begin{cases} \max\limits_{\substack{j \in \mathcal{I}_{\mathbf{N}_i} \\ j' \in \mathcal{I}_{\mathbf{N}_{i'}} \\ \gamma(j) = \gamma(j')}} \left\{ \frac{c^{\max}_j}{c^{\max}_{j'}} \right\}, & \forall i' \in \varphi(i) \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

$$\beta_i = \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_j \left(\sum_{j' \in \pi(j)} 1 \right) \tag{4.15}$$

we claim that the following inequality holds. We give its proof, together with the meaning of the coefficients α and β in Section 4.3.3:

$$
\begin{aligned}
\zeta_{i[k+1]}^{\max} &= \sum_{j\in\mathcal{I}_{\mathbf{N}_i}} c_j^{\max}\cdot\Big(\sum_{j'\in\pi(j)}\lceil q_{j'[k+1]}\rceil\Big)\\
&\le \sum_{i'\in\varphi(i)} \alpha_{ii'}\sigma_{i'[k+1]}^{\max}+\beta_i\\
&\le \sum_{i'\in\varphi(i)} \alpha_{ii'}(\sigma_{i'[k+1]}^{\max}+\zeta_{i'[k+1]}^{\max})+\beta_i\\
&\le \sum_{\substack{i'\in\varphi(i)\\ \sigma_{i'[k+1]}^{\max}\neq 0}} \alpha_{ii'}\Bigg(\sigma_{i'[k]}^{\max}+\zeta_{i'[k]}^{\max}+o_{i'[k+1]}^{\max}-o_{i'[k]}^{\max}\Bigg)+\\
&\quad \sum_{\substack{i'\in\varphi(i)\\ \sigma_{i'[k+1]}^{\max}\neq 0}} \alpha_{ii'}\Bigg((U_{i'[k]}^{\max}-1)(t_{[k+1]}-t_{[k]})\Bigg)+\beta_i
\end{aligned}
\tag{4.16}
$$

Plugging in the bound from (4.16) into inequality (4.13) gives us a working model of the system for predicting its worst-case behavior. When a resource $\mathbf{N}_i$ is starving, the evolution of its state depends on the other resources in the system which are non-starving ($\mathbf{N}_{i'}$, where $i'\in\varphi(i)$, and $\sigma_{i'[k+1]}^{\max}\neq 0$). We can thus rewrite inequation (4.13) for each resource $\mathbf{N}_i$ as having 2^n branches, each one representing a different combination of starving and non-starving resources. These inequations can then be aggregated into a matrix form as follows:

$$
\vec{x}_{[k+1]}\le A_{l_{[k]}}\Big(\vec{x}_{[k]}+(\vec{w}_{[k+1]}-\vec{w}_{[k]})+(\vec{u}_{[k]}-1_n)(t_{[k+1]}-t_{[k]})\Big)+\vec{b}_{l_{[k]}}
\tag{4.17}
$$

where $l:\mathcal{T}\to\{0,1,\cdots,2^n-1\}$ is a function associated with a running of the system, that maps time moments to the corresponding branches ($l_{[k]}$ is the according to which the system evolves from time $t_{[k]}$ onwards). The matrix $A_{l_{[k]}}$ and vector $\vec{b}_{l_{[k]}}$ are associated with the branch taken by the system at time $t_{[k]}$ and belong to the following sets:

$$
A_{l_{[k]}}\in\{A_0=0_{n\times n},A_1,\cdots,A_{2^n-2},A_{2^n-1}=I_n\}
$$

$$\begin{aligned}\vec{b}_{l_{[k]}} &\in \{\vec{b}_0, \vec{b}_1, \cdots, \vec{b}_{2^n-1} = 0_n\} \\ l_{[k]} &\in \{0, 1, \cdots, 2^n - 1\}\end{aligned}$$

The state, disturbance, and input to the system are:

$$\vec{x}_{[k]} = \left[\sigma^{\max}_{i[k]} + \zeta^{\max}_{i[k]}\right]_n \tag{4.18}$$

$$\vec{w}_{[k]} = \left[o^{\max}_{i[k]}\right]_n \tag{4.19}$$

$$\vec{u}_{[k]} = \left[U_{i[k]}\right]_n \tag{4.20}$$

At each moment in time $t_{[k]}$, the system evolves according to one of the 2^n branches $l_{[k]}$. The choice of the branch (and, implicitly, the corresponding matrix $A_{l_{[k]}}$ and vector $\vec{b}_{l_{[k]}}$) is unknown to the resource manager as we cannot tell at $t_{[k]}$ which resource will be starving at $t_{[k+1]}$. Such systems are called *switching systems with random switching*. The branch having A_{2^n-1} and $\vec{b}_{2^n-1}$ represents the case where all the resources are in non-starving mode. The branch having A_0 and $\vec{b}_0$ represents the case where all resources are starving, meaning all queues are empty.

In this section we have determined the worst-case behavior of an adaptive real-time system, We have done so starting from the general evolution of such systems described in Section 4.2. Inequation (4.12) describes an upper bound on the evolution of the accumulation of execution times on each resource in the system. These inequations describe the evolution of the resource in two cases: when starving and when non-starving. With the help of the approximation from inequality (4.16) we have managed to aggregate all these inequations (inequation (4.12) written for all resources in the system) into a matrix form (inequation (4.17)) that describes the worst-behavior evolution of the system. By worst-behavior we mean the behavior where the state of the system grows to its largest value. Our goal for the rest of the thesis is to take this worst-case behavior model (inequation 4.17) and determine conditions under which the state of the system remains bounded. Bounded state in the worst-case means that the system remains stable for all possible runtime scenarios, thus guaranteeing that all released jobs will eventually be executed.

4.3.1 System Model Parameters and Properties

We continue with a discussion on the form and properties of the A_l and $\vec{b_l}$, $l \in \{0, 1, \cdots, 2^n - 1\}$ matrices and vectors introduced in inequation (4.17). From inequation (4.13) we can observe that the matrices $A_l = [a_{lij}]$ and vectors $\vec{b_l} = [b_{li}]_n$ have a particular form. In any of the 2^n settings in which the system may find itself at a certain moment in time, all non-starving resources ($\mathbf{N}_i$) evolve depending only on themselves and, thus, the rows i in A_l have all elements 0, apart from a_{lii} which is 1. All starving resources i' evolve depending on the non-starving resources only and, thus, the rows i' in A_l have all elements associated with the starving resources 0. The elements associated with the non-starving resources on the rows i' are chosen from the coefficients α. For the vector $\vec{b_l}$ all elements associated with the non-starving resources are 0. The elements associated with the starving resources are taken from the factors β. For example, for a certain branch l where resources i to j are non-starving ($i < j$, $i, j \in \{1, 2, \cdots, n\}$), and resources 0 to $i - 1$ and $j + 1$ to n are starving, the matrix A_l and vector $\vec{b_l}$ have the form presented in Figure 4.3. Associated with these matrices we define the mapping:

$$S : \{A_0, \cdots A_{2^n-1}\} \to \{1, \cdots n\}$$

where $S(A_l)$ is the set of indexes of all non-starving resources (the indexes of all non-0 columns in A_l).

For each of the above matrices A_l and vectors b_l we have the following two properties:

1. all matrices A_l are idempotent: $A_l^2 = A_l$, $\forall l \in \{0, \cdots, 2^n - 1\}$, and

2. $A_l\vec{b_l} = 0_n$, $\forall l \in \{0, \cdots 2^n - 1\}$

$$
A_l = \begin{array}{c|ccccccccc|}
 & 1 & \cdots & i-1 & i & i+1 & \cdots & j & j+1 & \cdots & n \\
\hline
1 & 0 & \cdots & 0 & \alpha_{1,i} & \alpha_{1,i+1} & \cdots & \alpha_{1,j} & 0 & \cdots & 0 \\
2 & 0 & & 0 & \alpha_{2,i} & \alpha_{2,i+1} & & \alpha_{2,j} & 0 & & 0 \\
 & \vdots & & & \vdots & & & & \vdots & & \\
i-1 & 0 & & 0 & \alpha_{i-1,i} & \alpha_{i-1,i+1} & & \alpha_{i-1,j} & 0 & & 0 \\
i & 0 & & 0 & 1 & 0 & \cdots & 0 & 0 & & 0 \\
i+1 & 0 & & 0 & 0 & 1 & & 0 & 0 & & 0 \\
 & \vdots & & & \vdots & & \ddots & & \vdots & & \\
j & 0 & & 0 & 0 & 0 & & 1 & 0 & & 0 \\
j+1 & 0 & & 0 & \alpha_{j+1,i} & \alpha_{j+1,i+1} & \cdots & \alpha_{j+1,j} & 0 & & 0 \\
 & \vdots & & & \vdots & & & & \vdots & & \\
n & 0 & & 0 & \alpha_{n,i} & \alpha_{n,i+1} & & \alpha_{n,j} & 0 & & 0
\end{array}
\qquad
\vec{b_l} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{i-1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \beta_{j+1} \\ \vdots \\ \beta_n \end{pmatrix}
$$

Figure 4.3: The form of matrix A_l and vector $\vec{b_l}$ for the case when resources $\{1, \cdots, i-1, j+1, \cdots, n\}$ are starving and resources $\{i, i+1, \cdots, j\}$ are non-starving.

4.3.2 Proof of Inequality (4.12)

For any resource $\mathbf{N}_i$ and any time interval $[t_{[k]}, t_{[k+1]}]$, from equation (4.5) (and (4.7)) we have two cases to analyze. When $\sigma_{i[k+1]} = 0$ then inequality (4.12) obviously holds. When $\sigma_{i[k+1]} > 0$, we remember that during $[t_{[k]}, t_{[k+1]}]$ only jobs of one task $\tau_{j^\star}$ are being executed (see Section 4.1), therefore for all $j \in \mathcal{I}_{\mathbf{N}_i} \setminus \{j^\star\}$, equality

$$\begin{aligned} q_{j[k+1]} =& q_{j[k]} + \sum_{j' \in \pi(j)} (\lceil q_{j'[k]} \rceil - \lceil q_{j'[k+1]} \rceil) + \\ & \rho_{\gamma(j)[k]}(\phi_{\gamma(j)[k+1]} - \phi_{\gamma(j)[k]}) + \rho_{\gamma(j)[k]}(t_{[k+1]} - t_{[k]}) \end{aligned}$$

holds. We rewrite equation (4.5) as:

$$\begin{aligned} c_{j^\star[k]} q_{j^\star[k+1]} = c_{j^\star[k]} \Big(& q_{j^\star[k]} + \sum_{j' \in \pi(j^\star)} (\lceil q_{j'[k]} \rceil - \lceil q_{j'[k+1]} \rceil) + \\ & \rho_{\gamma(j^\star)[k]}(\phi_{\gamma(j^\star)[k+1]} - \phi_{\gamma(j^\star)[k]}) + \\ & \rho_{\gamma(j^\star)[k]}(t_{[k+1]} - t_{[k]}) \Big) - (t_{[k+1]} - t_{[k]}) \end{aligned}$$

and by replacing $c_{j^\star[k]}$ with $c_{j^\star}^{\max}$ we rewrite inequation (4.12) as:

$$\begin{aligned} c_{j^\star}^{\max} q_{j^\star[k+1]} \leq c_{j^\star}^{\max} \Big(& q_{j^\star[k]} + \sum_{j' \in \pi(j^\star)} (\lceil q_{j'[k]} \rceil - \lceil q_{j'[k+1]} \rceil) + \\ & \rho_{\gamma(j^\star)[k]}(\phi_{\gamma(j^\star)[k+1]} - \phi_{\gamma(j^\star)[k]}) + \\ & \rho_{\gamma(j^\star)[k]}(t_{[k+1]} - t_{[k]}) \Big) - (t_{[k+1]} - t_{[k]}) \end{aligned}$$

By rearranging the terms in the previous inequation we obtain:

$$\begin{aligned} c_{j^\star}^{\max} \Big(& q_{j^\star[k+1]} - q_{j^\star[k]} - \sum_{j' \in \pi(j^\star)} (\lceil q_{j'[k]} \rceil - \lceil q_{j'[k+1]} \rceil) - \\ & \rho_{\gamma(j^\star)[k]}(\phi_{\gamma(j^\star)[k+1]} - \phi_{\gamma(j^\star)[k]}) - \\ & \rho_{\gamma(j^\star)[k]}(t_{[k+1]} - t_{[k]}) \Big) \qquad \leq \qquad -(t_{[k+1]} - t_{[k]}) \end{aligned}$$

which holds for all moments of time, since all terms are positive values. Thus, inequation (4.12) holds for all moments of time and represents the worst-case behavior of the system. ■

4.3.3 Proof of Inequality (4.16)

The aim of this inequality is to bound the size of $\zeta^{\max}_{i[k+1]}$ ($\mathbf{N}_i$, $i \in \mathcal{I}_{\mathfrak{R}}$ being one of the resources of the system) relative to the states of the other resources in the system $\mathbf{N}_{i'}$, $i' \in \mathcal{I}_{\mathfrak{R}} \setminus \{i\}$.

Let us first deal with the first part of the inequality. By upper bounding all values $\lceil q_{j'[k+1]} \rceil$ with $q_{j'[k+1]} + 1$ we obtain the first part of the inequality:

$$\begin{aligned}\zeta^{\max}_{i[k+1]} &= \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_j \cdot \Bigg(\sum_{j' \in \pi(j)} \lceil q_{j'[k+1]} \rceil \Bigg) \\ &\leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_j \cdot \Bigg(\sum_{j' \in \pi(j)} q_{j'[k+1]} \Bigg) + \underbrace{\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c^{\max}_j \cdot \Bigg(\sum_{j' \in \pi(j)} 1 \Bigg)}_{\beta_i}\end{aligned} \tag{4.21}$$

We now wish to describe all quantities $q_{j'[k+1]}$ in terms of the maximum accumulation of execution times on their particular resources $\sigma^{\max}_{\nu(j')[k+1]}$. We observe the following inequality which expresses that queue $q_{j'}$ may, at most, contain all the accumulation of execution times on its respective resource:

$$q_{j'[k+1]} \leq \frac{\sigma^{\max}_{\nu(j')[k+1]}}{c^{\max}_{j'}}, \quad \forall j' \in \pi(j),\ j \in \mathcal{I}_{\mathbf{N}_i} \tag{4.22}$$

Plugging the above into inequality (4.21) we obtain:

$$\zeta^{\max}_{i[k+1]} \leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} \Bigg(\sum_{j' \in \pi(j)} \frac{c^{\max}_j}{c^{\max}_{j'}} \sigma^{\max}_{\nu(j')[k+1]} \Bigg) + \beta_i$$

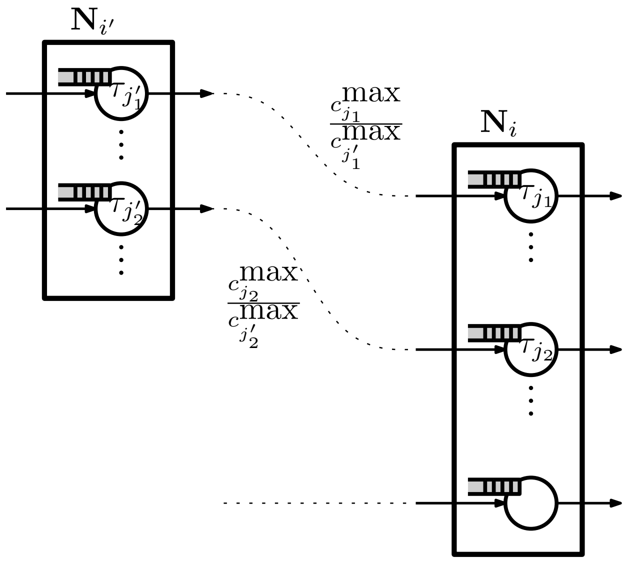


Figure 4.4: The case when two tasks on a resource have predecessors on a different resource.

which satisfies our goal. However, the above inequality gives us a very pessimistic upper bound of $\zeta^{\max}_{i[k+1]}$ since we may count each accumulation of execution times $\sigma^{\max}_{i'[k+1]}$, $i' \in \mathcal{I}_{\mathfrak{R}} \setminus \{i\}$ multiple times. To understand this, let us consider the example in Figure 4.4 where the resource $\mathbf{N}_i$ under analysis, has two tasks τ_{j_1} and τ_{j_2} that both have predecessors $\tau_{j'_1}$ and $\tau_{j'_2}$ respectively, running on resource $\mathbf{N}_{i'}$. In this case $\zeta^{\max}_{i[k+1]}$ is computed based on the sizes of the queues $q_{j'_1}$ and $q_{j'_2}$:

$$\zeta^{\max}_{i[k+1]} \leq \cdots + c^{\max}_{j_1} q_{j'_1[k+1]} + c^{\max}_{j_2} q_{j'_2[k+1]} + \cdots + \beta_i$$

By simply introducing the upper bound from inequality (4.22) we obtain:

$$\zeta^{\max}_{i[k+1]} \leq \cdots + \frac{c^{\max}_{j_1}}{c^{\max}_{j'_1}} \sigma^{\max}_{i'[k+1]} + \frac{c^{\max}_{j_2}}{c^{\max}_{j'_2}} \sigma^{\max}_{i'[k+1]} + \cdots + \beta_i$$

Here we count $\sigma^{\max}_{i'[k+1]}$ twice as we assume that this accumulation is fully contained in both jobs of $\tau_{j'_1}$ and jobs of $\tau_{j'_2}$ at the same time.

This approach is, of course, very pessimistic. A better approach is to observe that the worst case must happen when the whole accumulation on $\mathbf{N}_{i'}$ is split between these two tasks only: $\tau_{j'_1}$ and $\tau_{j'_2}$. The question remaining is how to determine the split between the two tasks. Since one of the coefficients: $\frac{c_{j_1}^{\max}}{c_{j'_1}^{\max}}$ and $\frac{c_{j_2}^{\max}}{c_{j'_2}^{\max}}$ is larger than the other, having all the accumulation on $\mathbf{N}_{i'}$ as jobs of the corresponding task give the worst case (largest) transfer of accumulation from $\mathbf{N}_{i'}$ to $\mathbf{N}_i$. For this example, a tight upper bound is thus found as follows:

$$\zeta_{i[k+1]}^{\max} \leq \cdots + \underbrace{\max\left\{\frac{c_{j_1}^{\max}}{c_{j'_1}^{\max}}, \frac{c_{j_2}^{\max}}{c_{j'_2}^{\max}}\right\}}_{\alpha_{ii'}} \sigma_{i'[k+1]}^{\max} + \cdots + \beta_i$$

Applying the idea presented above to inequality (4.21), and considering the definition of the α coefficients (equation (4.14)) we obtain:

$$\zeta_{i[k+1]}^{\max} \leq \sum_{i' \in \varphi(i)} \alpha_{ii'} \sigma_{i'[k+1]}^{\max} + \beta_i$$

which represents the first part of our inequality. We observe that $\sigma_{i'[k+1]}^{\max} \leq \sigma_{i'[k+1]}^{\max} + \zeta_{i'[k+1]}^{\max}$ since all these quantities are positive. By expanding this upper bound using inequation 4.12 (and inequation (4.13)) we obtain the second half of inequality (4.16) and, thus, complete the proof. ■

4.4 Illustrative Example

In Figure 4.5 we present an example of a system with two resources ($\mathbf{N}_1$ and $\mathbf{N}_2$) transversed by a task chain ($\mathfrak{C}$) of four tasks (τ_1, τ_2, τ_3, and τ_4), two on each resource (τ_1 and τ_4 on $\mathbf{N}_1$, and τ_2 and τ_3 on $\mathbf{N}_2$). The mapping of tasks to resources forms a cycle in the resource graph (a dependency from τ_1 on $\mathbf{N}_1$ to τ_2 on $\mathbf{N}_2$, and a dependency from τ_3 on $\mathbf{N}_2$ to τ_4 on $\mathbf{N}_1$). At each moment in time $t_{[k]}$, the state, disturbance, and input to the system (when modeling its worst-case

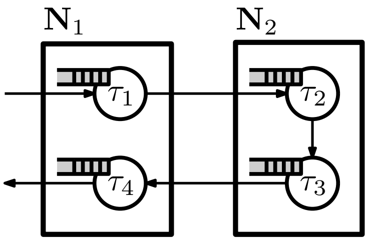


Figure 4.5: Example of a system with two resources and one task chain.

behavior) are (see equations (4.18), (4.19), and (4.20) on page 46):

$$\vec{x}_{[k]} = \begin{pmatrix} \underbrace{c_1^{\max} q_{1[k]} + c_4^{\max} q_{4[k]}}_{\sigma_{1[k]}^{\max}} + \underbrace{c_1^{\max} 0 + c_4^{\max}(\lceil q_{1[k]} \rceil + \lceil q_{2[k]} \rceil + \lceil q_{3[k]} \rceil)}_{\zeta_{1[k]}^{\max}} \\ \underbrace{c_2^{\max} q_{2[k]} + c_3^{\max} q_{3[k]}}_{\sigma_{1[k]}^{\max}} + \underbrace{c_2^{\max} \lceil q_{1[k]} \rceil + c_3^{\max}(\lceil q_{1[k]} \rceil + \lceil q_{2[k]} \rceil)}_{\zeta_{2[k]}^{\max}} \end{pmatrix}$$

$$\vec{w}_{[k]} = \begin{pmatrix} c_1^{\max} \rho_{[k-1]} \phi_{1[k]} + c_4^{\max} \rho_{[k-1]} \phi_{4[k]} \\ c_2^{\max} \rho_{[k-1]} \phi_{2[k]} + c_3^{\max} \rho_{[k-1]} \phi_{3[k]} \end{pmatrix}$$

$$\vec{u}_{[k]} = \begin{pmatrix} (c_1^{\max} + c_4^{\max}) \rho_{[k]} \\ (c_2^{\max} + c_3^{\max}) \rho_{[k]} \end{pmatrix}$$

The coefficients α and β for this system are:

$$\alpha_{12} = \max\left\{ \frac{c_4^{\max}}{c_3^{\max}}, \frac{c_4^{\max}}{c_2^{\max}} \right\} \qquad \alpha_{21} = \max\left\{ \frac{c_2^{\max}}{c_1^{\max}}, \frac{c_3^{\max}}{c_1^{\max}} \right\}$$

$$\beta_1 = 3c_4^{\max} \qquad \beta_2 = c_2^{\max} + 2c_3^{\max}$$

This system has $2^2 = 4$ branches according to which it can evolve, with:

$$A_l \in \left\{ \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}}_{A_0}, \underbrace{\begin{pmatrix} 1 & 0 \\ \alpha_{21} & 0 \end{pmatrix}}_{A_1}, \underbrace{\begin{pmatrix} 0 & \alpha_{12} \\ 0 & 1 \end{pmatrix}}_{A_2}, \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{A_3} \right\} \text{ and}$$

$$\vec{b}_l \in \left\{ \underbrace{\begin{pmatrix}\beta_1\\ \beta_2\end{pmatrix}}_{\vec{b}_1}, \underbrace{\begin{pmatrix}0\\ \beta_2\end{pmatrix}}_{\vec{b}_2}, \underbrace{\begin{pmatrix}\beta_1\\ 0\end{pmatrix}}_{\vec{b}_3}, \underbrace{\begin{pmatrix}0\\ 0\end{pmatrix}}_{\vec{b}_4} \right\}.$$

4.5 Understanding the System Model

We shall now give an intuitive, informal discussion about the functioning of our system. The exact worst-case behavior of the system is given by inequation (4.12) (also written as (4.13)). An upper bound to this behavior was determined in inequation (4.17) which represents our worst-case system model. Let us start with inequation (4.13) and describe its parameters and its functioning. Figure 4.6 presents the parameters associated with the behavior of resource $\mathbf{N}_i$ in between time moments $t_{[k]}$ and $t_{[k+1]}$. $\sigma_{i[k]}^{\max}$ represents the total accumulation of execution time (assuming worst-case execution times) associated with the jobs existing at time $t_{[k]}$ in the queues of the tasks running on $\mathbf{N}_i$. $\zeta_{i[k]}^{\max}$ is the accumulation of execution times of all released, but not yet ready for execution jobs of tasks running on $\mathbf{N}_i$. These are the jobs in the queues of all tasks that precede the tasks on $\mathbf{N}_i$ on their respective task chains. At a given moment of time $t_{[k]}$ the state of the system is $\vec{x}_{i[k]} = \sigma_{i[k]}^{\max} + \zeta_{i[k]}^{\max}$ and represents the maximum amount of execution time that $\mathbf{N}_i$ has to process in order to finish executing all currently released jobs. If the currently released jobs have execution times smaller than their worst-case execution times, the accumulation is smaller and can be executed faster. Hence the less-or-equal sign in equation (4.13) which described the evolution of $\vec{x}_{i[k]}$.

The amount $U_{i[k]}^{\max}(t_{[k+1]} - t_{[k]}) + o_{i[k+1]}^{\max} - o_{i[k]}^{\max}$ represents the amount of execution time, meant for $\mathbf{N}_i$, added into the system between $t_{[k]}$ and $t_{[k+1]}$. We prefer writing the incoming accumulation in this form because $U_{i[k]}$ represents the worst-case load (computed assuming worst-case execution times) that $\mathbf{N}_i$ should be subjected to during $[t_{[k]}, t_{[k+1]}]$. The exact load ($U_{i[k]}$) added into the system,

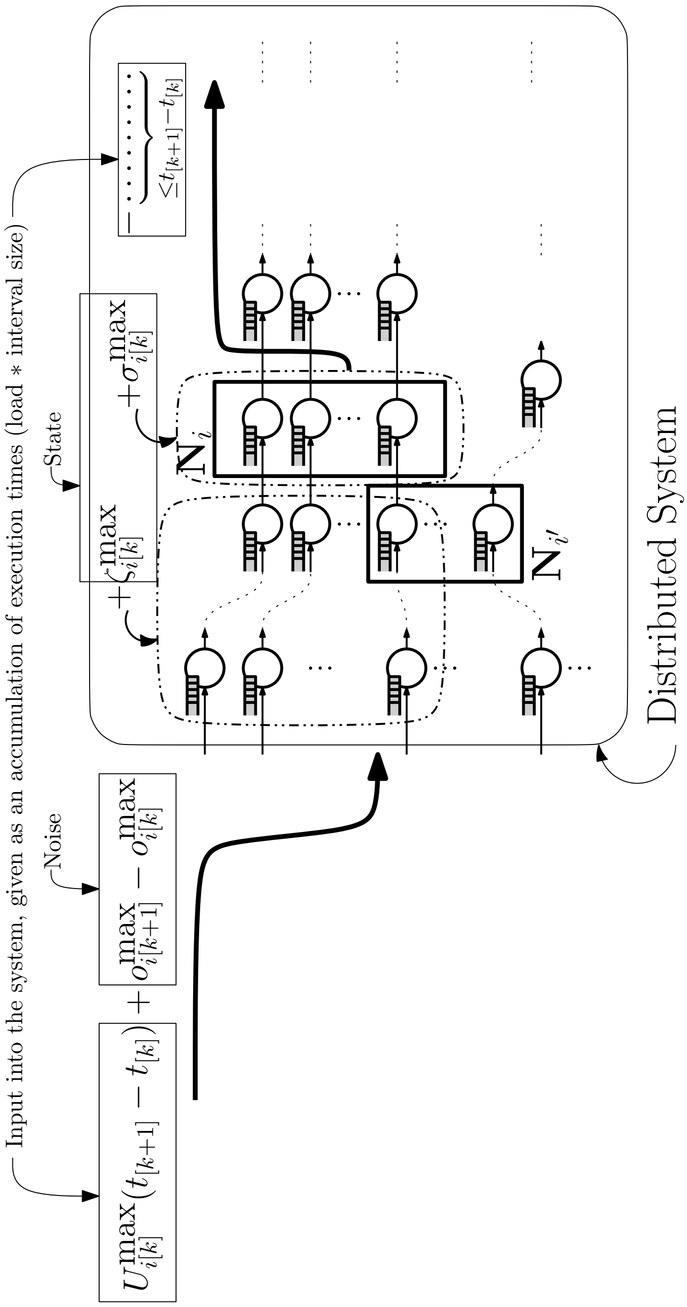

Input into the system, given as an accumulation of execution times (load * interval size)
Noise
State
$U^{\max}_{i[k]}(t_{[k+1]} - t_{[k]})$
$+\, o^{\max}_{i[k+1]} - o^{\max}_{i[k]}$
$+\varsigma^{\max}_{i[k]}$
$+\sigma^{\max}_{i[k]}$
$\leq t_{[k+1]} - t_{[k]}$
N_i
$\mathrm{N}_{i'}$
Distributed System

Figure 4.6: The parameters of the system model.

for each resource, should be less than one as the jobs will become unschedulable on the resource otherwise. At times, the resource manager may wish to decrease the queue sizes and, in order to guarantee that, it may select task chain rates such that $U_{i[k]}^{\max} \leq 1$.

The amount $t_{[k+1]} - t_{[k]}$ represents the maximum amount of execution time removed from $\vec{x}_i$ during $[t_{[k]}, t_{[k+1]}]$. New jobs enter as an accumulation of execution times in the $\zeta_i^{\max}$ part of the state. From there they move to $\sigma_i^{\max}$ where they can be executed and removed from the state.

When $\mathbf{N}_i$ is non-starving, the $\sigma_i^{\max}$ part of the state is sufficiently large to allow the resource to execute jobs continuously, without idling, during $[t_{[k]}, t_{[k+1]}]$. In this case the evolution of the state of the system, in the worst-case when the execution times of all jobs are at their largest, is easy to describe, since we know precisely how much accumulation enters and leaves the resource.

When $\mathbf{N}_i$ is starving $\sigma_i^{\max} = 0$ and no jobs get executed during $[t_{[k]}, t_{[k+1]}]$ and we know the rate at which $\zeta_i^{\max}$ is increasing. But is there any bound on the growth of $\zeta_i^{\max}$ or not? To answer this question we remember that $\zeta_i^{\max}$ is formed of the queues of jobs of the tasks that are predecessors, in their respective task chains, to the tasks running on $\mathbf{N}_i$. Not all queues are empty and thus, the resources running the tasks that have non-empty task queues (such as $\mathbf{N}_{i'}$ in Figure 4.6) are not starving. All such resources $\mathbf{N}_{i'}$ are executing jobs, but these jobs must be jobs of tasks which do not have successors on $\mathbf{N}_i$, otherwise these jobs will keep $\mathbf{N}_i$ from idling. At some point, however these resources (that run tasks predecessor to the ones on $\mathbf{N}_i$) will deplete all their other queues and then they must begin executing jobs that will reach $\mathbf{N}_i$ and move it into a non-starving state. This suggests that the bound of the increase of $\zeta_i^{\max}$, when $\mathbf{N}_i$ is starving, is given by the worst-case amount of execution times ($\sigma_{i'}^{\max}$) existing in the other non-starving resources in the system. To compute the accumulation of execution times, we have assumed for all jobs that will reach $\mathbf{N}_i$ the worst-case execution times associated with their tasks running on $\mathbf{N}_i$. Similarly, we do so for all other resources in the system. When computing the bound on $\zeta_i^{\max}$ (when $\mathbf{N}_i$ is starving) we must rewrite the accumulations on the other resources in terms of the execution times of the tasks running on $\mathbf{N}_i$. We do so by multiplying them with the coefficients α described in equation (4.16) since they describe the worst-case accumulation that is added to $\sigma_i^{\max}$ when a job moves from another resource into $\mathbf{N}_i$.

We conclude this section with a discussion on the potential pessimism characteristic to the worst-case system model derived in Section 4.3. The model presented in equation (4.13) is an accurate description of the worst-case behavior of the system, since the situation described there corresponds with all jobs of all tasks executing with their worst-case execution time. However, as explained in Section 4.3, this model is indeterminate when a resource starves. We fix that by developing equation (4.17) which introduces pessimism into the model due to the upper bounding done in inequality (4.16):

$$\zeta_{i[k+1]}^{\max} \leq \sum_{i' \in \varphi(i)} \alpha_{ii'} \sigma_{i'[k+1]}^{\max} + \beta_i \leq \sum_{i' \in \varphi(i)} \alpha_{ii'} (\sigma_{i'[k+1]}^{\max} + \zeta_{i'[k+1]}^{\max}) + \beta_i$$

The pessimism appears because we consider $\sigma_{i'[k+1]}^{\max} + \zeta_{i'[k+1]}^{\max}$ instead of $\sigma_{i'[k+1]}^{\max}$. We present the effects of the pessimism with the help of Figure 4.7 where for resource $\mathbf{N}_5$ the quantity $\zeta_5^{\max}$ is approximated in the following way:

$$\begin{aligned}
\zeta_5^{\max} =& c_9^{\max}(\lceil q_3 \rceil + \lceil q_6 \rceil) + c_{10}^{\max}(\lceil q_4 \rceil + \lceil q_7 \rceil) \\
\leq& \Big(c_9^{\max}(q_3 + q_6) + c_{10}^{\max}(q_4 + q_7) \Big) + \underbrace{(2c_9^{\max} + 2c_{10}^{\max})}_{\beta_5} \\
=& \Bigg(\Big(\frac{c_9^{\max}}{c_3^{\max}} c_3^{\max} q_3 + \frac{c_9^{\max}}{c_6^{\max}} c_6^{\max} q_6 \Big) + \\
& \Big(\frac{c_{10}^{\max}}{c_4^{\max}} c_4^{\max} q_4 + \frac{c_{10}^{\max}}{c_7^{\max}} c_7^{\max} q_7 \Big) \Bigg) + \beta_5
\end{aligned}$$

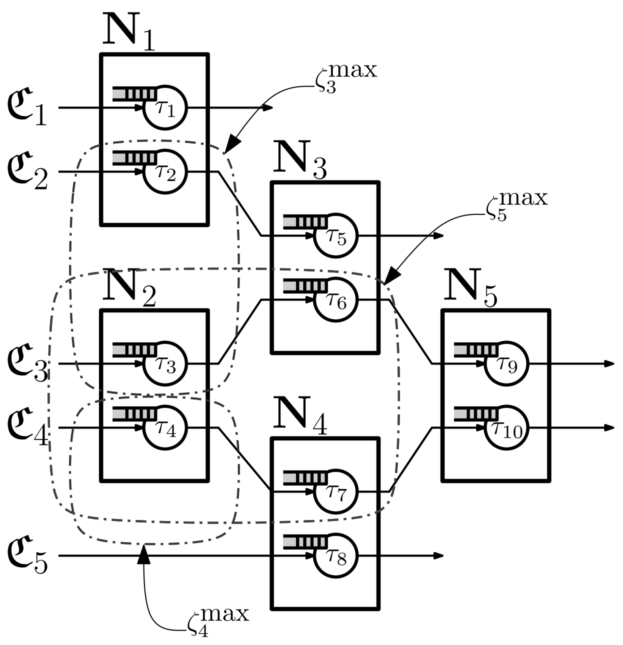


Figure 4.7: Example illustrating the pessimism of the method.

$$\leq \underbrace{\max\left\{\frac{c_9^{\max}}{c_3^{\max}}, \frac{c_{10}^{\max}}{c_4^{\max}}\right\}}_{\alpha_{52}} \sigma_2^{\max} + \underbrace{\frac{c_9^{\max}}{c_6^{\max}}}_{\alpha_{53}} \sigma_3^{\max} + \underbrace{\frac{c_{10}^{\max}}{c_7^{\max}}}_{\alpha_{54}} \sigma_4^{\max} + \beta_5$$

$$\leq \Bigg(\alpha_{52}(\sigma_2^{\max} + \underbrace{0}_{\zeta_2^{\max}}) + \alpha_{53}\Big(\sigma_3^{\max} + \underbrace{c_6^{\max}\lceil q_3 \rceil + c_5^{\max}\lceil q_2 \rceil}_{\zeta_3^{\max}}\Big) +$$

$$\alpha_{54}\Big(\sigma_4^{\max} + \underbrace{c_7^{\max}\lceil q_4 \rceil}_{\zeta_4^{\max}}\Big) \Bigg) + \beta_5$$

We can see from the figure and from the above computation that by including $\alpha_{53}\zeta_3^{\max} + \alpha_{54}\zeta_4^{\max}$ we end up counting the jobs of $\mathbf{N}_i$ twice ($\alpha_{52}\sigma_2^{\max}$), thus introducing pessimism.

The amount of pessimism increases with longer task chains because they pass through more resources $\mathbf{N}_{i'}$, $i' \in \varphi(i)$ before arriving at the resource of interest $\mathbf{N}_i$ and, thus, including more pessimistic approximations of the form $\sigma_{i'}^{\max} \leq \sigma_{i'}^{\max} + \zeta_{i'}^{\max}$.

The consequences of the above pessimism are twofold: firstly the pessimism affects the state of the system making it appear larger than it is. This is important as the largest state of the system helps us determine bounds on the real-time properties such as worst-case response times (larger bounds for larger state, see Section 6.3). Secondly the pessimism affects the system stability property where, as we shall see in Section 5.2, the sufficient condition for a system to have stable resource managers depends on the coefficients of the A_l, $l \in \{0, 1, \cdots 2^n - 1\}$ matrices and smaller coefficients generally imply better stability properties.

5 Stability Conditions

Up to this point we have determined that our distributed real-time system can be modeled as a switching system with random switching. This system evolves according to one of 2^n possible scenarios and, at every moment in time, the scenario may switch to a different one. As described in Section 3.4, the system is stable when a norm of its state ($|\vec{x}_{[k]}|$) remains bounded at all times, regardless of the switching pattern the system may employ.

At runtime, the system is controlled by its resource manager. For a given system we may envision a set of resource manager policies that are able to keep the system stable. These policies may differ amongst themselves by their performance[1] but they are all capable of keeping the system stable.

In this chapter we wish to determine whether this set of resource manager policies is empty or not. If this set is empty, the system

[1]We do not address the performance of a certain resource manager policy in this work, as performance may mean different things for different systems and applications.

cannot be kept stable (assuming its worst-case behavior) regardless of the resource manager policy used. To this end, we develop a set of conditions that, if satisfied by the system parameters, guarantee the existence of resource manager policies that can keep the system stable.

The developed conditions will be conditions on the parameters of the system such as the task chain rate sets, the tasks' execution time intervals, and the topology of the system (the mapping of tasks to resources and the interdependencies between resources). These conditions are not conditions on the resource manager. The conditions come in two flavors:

- *Necessary conditions*: if not satisfied, there is no resource manager policy that can stabilize the system,
- *Sufficient conditions*: if satisfied, there certainly exist resource manager policies that can stabilize the system.

Finally, in Section 5.3 we derive the conditions under which a certain resource manager is guaranteed to keep the system stable.

5.1 Necessary Condition for Stability

Theorem 1:
A necessary condition for a system to be stable is:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(j)}^{\min} \leq 1, \; \forall i \in \mathcal{I}_{\mathfrak{R}} \tag{5.1}$$

◇

Proof We prove this by contradiction. We allow equation (5.1) not to hold and we will show that this leads to an unstable system. If equation (5.1) does not hold, there exists at least one resource $\mathbf{N}_i$ that will have:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(\tau_j)}^{\min} - 1 = \beta_i > 0$$

For the worst-case evolution $c_{j[k]} = c_j^{\max}$, $\forall j \in \mathcal{I}_{\mathbf{N}_i}$, $\forall k$ we can rewrite equation (4.12) as:

$$\sigma_{i[k+1]}^{\max} + \zeta_{i[k+1]}^{\max} = \\ \max\Big\{ \zeta_{i[k+1]}^{\max}, \quad \sigma_{i[k]}^{\max} + \zeta_{i[k]}^{\max} - o_{i[k]}^{\max} + o_{i[k+1]}^{\max} + \beta_i(t_{[k+1]} - t_{[k]}) \Big\}$$

It is easy to see from above that:

$$\sigma_{i[k+1]}^{\max} + \zeta_{i[k+1]}^{\max} \geq \sigma_{i[k]}^{\max} + \zeta_{i[k]}^{\max} - o_{i[k]}^{\max} + o_{i[k+1]}^{\max} + \beta_i(t_{[k+1]} - t_{[k]}), \quad \forall k.$$

Since $o_{i[k+1]}^{\max}$ is bounded

$$0 \leq o_{i[k+1]}^{\max} \leq \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max}, \quad \forall k$$

we have that

$$\lim_{k\to\infty} \sigma_{i[k+1]}^{\max} + \zeta_{i[k+1]}^{\max} = \infty$$

which implies that there exists at least one task $\tau_{j'}$, where $j' \in \mathcal{I}_{\mathbf{N}_i} \cup \Big(\bigcup_{j \in \mathcal{I}_{\mathbf{N}_i}} \pi(j) \Big)$ such that $q_{j'[k]} \xrightarrow{k\to\infty} \infty$. It then follows that $|\vec{x}_{[k]}| \xrightarrow{k\to\infty} \infty$ and the proof is complete. ■

If theorem 1 holds, then the set $\Gamma_\star$, defined below, is not empty:

$$\Gamma_\star = \Big\{ \vec{\rho}^\star \in \mathbf{P} \,\Big|\, \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \cdot \rho_{\gamma(\tau_j)}^\star \leq 1, \forall i \in \mathcal{I}_{\mathfrak{R}} \Big\} \neq \emptyset \tag{5.2}$$

The $\Gamma_\star$ set is important as it describes all other rate vectors (apart from $\vec{\rho}^{\min}$) that can be used to satisfy Theorem 1. If the system satisfies the sufficient conditions presented in the next section, using any rate vector from $\Gamma_\star$ guarantees that, under any execution time variations, all resources can handle the amount of execution time generated for them by the released jobs in the system.

5.1.1 Example of an Unstable System

We shall now present an example showing that, while condition (5.1) is necessary, it is not sufficient. The example presented here is a variation of the classic Lu-Kumar network (see [Bra08] chapter 3) from the domain of multi-class queueing networks.

Let us consider the system from Figure 4.5 for which we have $c_1^{\max} = c_3^{\max} = 1/5$, $c_2^{\max} = c_4^{\max} = 3/5$ and the input rate is $\rho^{\min} = 1$. All these numbers are given in appropriate time units. This system obviously satisfies Theorem 1.

Let us assume that the schedulers on each resource are fixed priority based where τ_4 and τ_2 have the highest priority on their respective resources. Let us also assume that the resource manager always chooses the minimum task chain release rate $\rho^{\min}$ and the system evolves such that execution times are always at their maximum. If at time $t = 0$, $q_1 = 2N$ and all other queues are empty, then the evolution in time of the system is as shown in Table 5.1. At time $t = 1/5$ a job of τ_1 gets executed and moves to the queue of q_2. From this point on tasks τ_1 on $\mathbf{N}_1$ and τ_2 on $\mathbf{N}_2$ execute jobs in parallel. The jobs executed from q_2 are moved to q_3, but because τ_2 has priority over τ_3, $\mathbf{N}_2$ will execute jobs from q_3 only after q_2 becomes empty. Because the execution time of jobs of τ_1 is smaller than the execution time of jobs of τ_2, jobs from q_1 move to q_2 faster than from q_2 to q_3, thus $\mathbf{N}_2$ only begins executing jobs of τ_3 when the initial jobs in the system have been processed by τ_1 and τ_2. During the execution of these jobs, new jobs arrive into the system and move from q_1 to q_2. Considering the rate at which jobs arrive into the system (1 job every 1 time unit) the queue of τ_2 will finally become empty at time $t = 3N + 4/5$ time units. At this time the queue of τ_3 will now contain $q_3 = 5N + 1$ jobs. One job from q_3 gets executed until $t = 3N + 1$ time units and moves to q_4. From this point on, τ_3 and τ_4 execute jobs in parallel until q_3 empties (because jobs of τ_3 have smaller execution times than jobs of τ_4). On $\mathbf{N}_1$, τ_4 has priority over τ_1 so, while q_4 empties, jobs accumulate on q_1. τ_4 has to execute $5N + 1$ jobs, so they will be finished in

Table 5.1: Possible behavior of the system presented in Figure 4.5.

t	0	$\frac{1}{5}$	$3N+\frac{4}{5}$	$3N+1$	$6N+\frac{8}{5}$
q_1	$2N$	$2N-1$	0	1	$3N+1$
q_2	0	1	0	0	0
q_3	0	0	$5N+1$	$5N$	0
q_4	0	0	0	1	0

$3/5 * 5N + 3/5$ time units. During this time $3N+1$ jobs accumulate in q_1. We can observe that at $t = 6N + 8/5$ the state of the system is similar with the one at $t = 0$ in the sense that queues q_2, q_3 and q_4 are empty but the size of q_1 has increased from $2N$ to $3N+1$ jobs. Since N has been chosen arbitrarily, we can see that the behavior of the system is cyclic and the sizes of q_1 increases progressively with every cycle and, thus, the system is unstable, even though Theorem 1 is satisfied.

5.2 Sufficient Condition for Stability

In this section we augment the conditions derived above with sufficient conditions that, if satisfied by a system, guarantee that the state of the system reduces when rates from $\Gamma_\star$ are applied by the resource manager. The conditions that we will derive emerge from the topology (the mapping of tasks chains to resources) and parameters (execution time and rate intervals) of the system. They are not conditions on the behavior of the resource manager.

In this section we will show that the system is stable when subjected to an un-controlled but bounded input. Here we treat as inputs both the input $\vec{u}_{[k]}$ (which is bounded to all possible values generated by the rate vectors in $\Gamma_\star$) and the perturbation $\vec{w}_{[k]}$ (which is also bounded: $0_n \preceq \vec{w}_{[k]} \preceq \vec{w}^{\max} = [\vec{w}_i^{\max}]_n$ where $\vec{w}_i^{\max} = \sum\limits_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max}$). By stability, we mean uniform boundedness (see Section 3.4) where the norm of the state decreases, in time, from its initial value $|\vec{x}_{[0]}|$ down below some non-0 bound [Mic08].

Theorem 2:
For each matrix $A_l = [a_{l\,kp}]_{n\times n}$ from (4.17) we define the matrix $B_l = [b_{l\,kp}]_{n\times n}$ as:

$$b_{l\,kp} = \begin{cases} \mu, & \text{if } k = p \\ -a_{l\,kp}, & \text{otherwise} \end{cases}; \quad \text{where } k, p \in \{1, \cdots, n\};\ 0 < \mu < 1$$

For a system that satisfies Theorem 1 the system modeled in equation (4.17) can be stabilized using rates from the set $\Gamma_\star$ if there exits a point $\vec{p} = [p_i]_n$, $p_i > 0$, $p_{n+1} > 0$, and some $0 < \mu < 1$ such that:

$$\begin{pmatrix} B_l & -b_l \\ 0, 0, \cdots 0 & \mu \end{pmatrix} \begin{pmatrix} \vec{p} \\ p_{n+1} \end{pmatrix} \succeq 0, \quad \forall l \in \{1, \cdots, 2^n\} \tag{5.3}$$

◇

Proof We construct the proof in three steps:

1. We construct a norm $|\cdot|_p$ for the system:
$$\vec{x}_{[k+1]} = A_{l_{[k]}} \vec{x}_{[k]} \tag{5.4}$$
such that:
$$|\vec{x}_{[k+1]}|_p \leq |\vec{x}_{[k]}|_p, \quad \forall k \in \mathbb{Z}_+ \tag{5.5}$$
where the matrices $A_{l_{[k]}}$ are the same as in equation (4.17).

2. We show that using the above determined norm, the system
$$\vec{x}_{[k+1]} = A_{l_{[k]}} \Big(\vec{x}_{[k]} + (\vec{w}_{[k+1]} - \vec{w}_{[k]}) + (\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]}) \Big) \tag{5.6}$$
which evolves such that $\vec{x}_{[k]} \succeq 0_n$, $\forall k \in \mathbb{Z}_+$ is GAS with respect to $\vec{u}$ and ISS with respect to $\vec{w}$.

3. We extend the above discussion to the model of our system presented in equation (4.17)

Step 1. We claim that the norm

$$|\vec{x}|_p = \max_{i \in \mathcal{I}_{\mathfrak{R}}} \left\{ \frac{|x_i|}{p_i} \right\}, \vec{p} = [p_i]_n, p_i > 0 \tag{5.7}$$

that satisfies:

$$B_l \vec{p} \succeq 0_n, \quad \forall l \in \{1, 2, \cdots, 2^n\} \tag{5.8}$$

for some $0 < \mu < 1$ is a norm that satisfies condition (5.5). To prove this, let us show that if there exits $\vec{p}$ such that $B_l \vec{p} \succeq 0_n$ for some l than $|A_l \vec{x}|_p \leq |\vec{x}|_p$ for any $\vec{x}$. Remembering that with $S(A_l)$ we denote the set of non-starving resources when the system evolves according with branch l, let us observe that:

$$B_l \vec{p} \succeq 0_n \Rightarrow \begin{cases} \mu p_i - \sum\limits_{i' \in S(A_l)} a_{lii'} p_{i'} \geq 0, & \forall i \in \mathcal{I}_{\mathfrak{R}} \setminus S(A_l) \\ \mu p_{i'} \geq 0, & \forall i' \in S(A_l) \end{cases}$$
$$\Rightarrow \sum_{i' \in S(A_l)} a_{lii'} \frac{p_{i'}}{p_i} \leq \mu < 1, \quad \forall i \in \mathcal{I}_{\mathfrak{R}} \setminus S(A_l)$$

We then have:

$$|A_l \vec{x}|_p = \max \left\{ \max_{i \in \mathcal{I}_{\mathfrak{R}} \setminus S(A_l)} \left\{ \frac{\left| \sum\limits_{i' \in S(A_l)} a_{lii'} x_{i'} \right|}{p_i} \right\}, \max_{i' \in S(A_l)} \left\{ \frac{|x_{i'}|}{p_{i'}} \right\} \right\} \tag{5.9}$$

For all $\forall i \in \mathcal{I}_{\mathfrak{R}} \setminus S(A_l)$ we have the following inequality:

$$\begin{aligned} \frac{\left| \sum\limits_{i' \in S(A_l)} a_{lii'} x_{i'} \right|}{p_i} &\leq \sum_{i' \in S(A_l)} a_{lii'} \frac{p_{i'}}{p_i} \frac{|x_{i'}|}{p_{i'}} \\ &\leq \frac{1}{\mu} \sum_{i' \in S(A_l)} a_{lii'} \frac{p_{i'}}{p_i} \frac{|x_{i'}|}{p_{i'}} \leq \frac{\sum\limits_{i' \in S(A_l)} a_{lii'} \frac{p_{i'}}{p_i} \frac{|x_{i'}|}{p_{i'}}}{\sum\limits_{i' \in S(A_l)} a_{lii'} \frac{p_{i'}}{p_i}} \\ &\leq \max_{i' \in S(A_l)} \left\{ \frac{|x_{i'}|}{p_{i'}} \right\} \leq |\vec{x}|_p \end{aligned} \tag{5.10}$$

and thus $|A_l\vec{x}|_p \leq |\vec{x}|_p$. If there exists a point $\vec{p}$ such that condition (5.8) is satisfied, then norm (5.7) is a norm that satisfies condition (5.5) for system (5.4).
Step 2. We recursively expand equation (5.6) as:

$$\vec{x}_{[k+1]} = A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}} \vec{x}_{[0]} + \vec{U}_{S[k]} + \vec{\Omega}_{S[k]}$$

where:

$$\begin{aligned}\vec{U}_{S[k]} =& A_{l_{[k]}}(\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]}) + \\ & A_{l_{[k]}} A_{l_{[k-1]}}(\vec{u}_{[k-1]} - 1_n)(t_{[k]} - t_{[k-1]}) + \\ & \cdots + A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}}(\vec{u}_{[0]} - 1_n)(t_{[1]} - t_{[0]}) \\ \vec{\Omega}_{S[k]} =& A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]}) + A_{l_{[k]}} A_{l_{[k-1]}}(\vec{w}_{[k]} - \vec{w}_{[k-1]}) + \\ & \cdots + A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}}(\vec{w}_{[1]} - \vec{w}_{[0]})\end{aligned}$$

Let us observe that $\vec{\rho}_{[k]} \in \Gamma_\star$ implies that $\vec{u}_{[k]} \preceq 1_n$ (see equations (4.11) and 4.20) which implies that $\vec{U}_{S[k]} \preceq 0_n$. We then have:

$$|\vec{x}_{[k+1]}| \leq |A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}} \vec{x}_{[0]} + \vec{U}_{S[k]}|_p + |\vec{\Omega}_{S[k]}|_p$$

Since $\vec{x}_{[k]} \succeq 0_n$, $\forall k \in \mathbb{Z}_+$ we have that:

$$A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}} \vec{x}_{[0]} \succeq A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}} \vec{x}_{[0]} + \vec{U}_{S[k]} \succeq -\vec{\Omega}_{S[k]}$$

and thus:

$$|\vec{x}_{[k+1]}| \leq \max\{|\vec{\Omega}_{S[k]}|_p, |A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}} \vec{x}_{[0]}|_p\} + |\vec{\Omega}_{S[k]}|_p$$

We want to show that the sequence $\left\{|\vec{\Omega}_{S[k]}|_p\right\}_k$ is bounded. We observe that:

$$\begin{aligned}\vec{\Omega}_{S[k]} =& A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]}) + A_{l_{[k]}} \vec{\Omega}_{S[k-1]} \\ =& A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]}) + A_{l_{[k]}} A_{l_{[k-1]}}(\vec{w}_{[k]} - \vec{w}_{[k-1]}) + \\ & A_{l_{[k]}} A_{l_{[k-1]}} \vec{\Omega}_{S[k-2]}\end{aligned}$$

$$= \cdots$$

Let us analyze the behavior of the sequence in two extreme cases:

1. if the matrices $A_{l_{[k]}}$ are such that $S(A_{l_{[k]}}) \subseteq S(A_{l_{[k-1]}})$, $\forall k \in \mathbb{Z}_+$, then one can directly verify that $A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[1]}} = A_{l_{[k]}}$. Then we have:

$$\begin{aligned}\vec{\Omega}_{S[k]} &= A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[k]} + \vec{w}_{[k]} - \vec{w}_{[k-1]} + \cdots - \vec{w}_{[0]}) \\ &= A_{l_{[k]}}(\vec{w}_{[k+1]} - \vec{w}_{[0]})\end{aligned}$$

   and it follows that:

$$|\vec{\Omega}_{S[k]}|_p \leq |\vec{w}^{\max}|_p, \quad \forall k \in \mathbb{Z}_+$$

2. if the matrices $A_{l_{[k]}}$ are such that $S(A_{l_{[k]}}) \cap S(A_{l_{[k-1]}}) = \emptyset$, $\forall k \in \mathbb{Z}_+$, then one can verify – by successive application of norm (5.7) to $\vec{x}$, $A_{l_{[k]}}\vec{x}$, $A_{l_{[k]}}A_{l_{[k-1]}}\vec{x}$, ...as in (5.9) and by repeating the reasoning from inequality (5.10) – that $|A_{l_{[k]}}A_{l_{[k-1]}}\vec{x}|_p \leq \mu|\vec{x}|_p$, $|A_{l_{[k]}}A_{l_{[k-1]}}A_{l_{[k-2]}}|_p \leq \mu^2|\vec{x}|_p$, ..., $\forall \vec{x}$ and then we have the following inequality:

$$|\vec{\Omega}_{S[k]}| \leq |\vec{w}^{\max}|_p \sum_{k'=1}^{k} \mu^{k'-1} \leq \frac{|\vec{w}^{\max}|_p}{1-\mu}, \quad \forall k \in \mathbb{Z}_+$$

In the general case, while a resource is non-starving, the component of $\vec{\Omega}_{S[k]}$ associated with it behaves as in the first case (we can observe this from inequation (4.13) upon which our model is based). Meanwhile, the components of $\vec{\Omega}_{S[k]}$ associated with starving resources behave as in the second case (we can observe that from inequality (5.10)). Finally we have that:

$$|\vec{\Omega}_{S[k]}|_p \leq \max\{|\vec{w}^{\max}|_p, \frac{|\vec{w}^{\max}|_p}{1-\mu}\} = \frac{|\vec{w}^{\max}|_p}{1-\mu}, \quad \forall k \in \mathbb{Z}_+$$

and thus the sequence $\{|\vec{\Omega}_{S[k]}|_p\}_k$ is bounded.

Since $|\vec{\Omega}_{S[k]}|_p \leq \frac{|\vec{w}^{\max}|_p}{1-\mu}$ we have that the system captured by equation (5.6) is GAS with respect to $\vec{w}$ and ISS with respect to $\vec{u}$, and thus stable.

Step 3. Our system model (4.17) may be rewritten in homogeneous coordinates as:

$$\begin{pmatrix} \vec{x}_{[k+1]} \\ 1 \end{pmatrix} = \begin{pmatrix} A_{l_{[k]}} & \vec{b}_{l_{[k]}} \\ 0, \cdots 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{x}_{[k]} \\ 1 \end{pmatrix} + \begin{pmatrix} A_{l_{[k]}} & \vec{b}_{l_{[k]}} \\ 0, \cdots 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{w}_{[k+1]} - \vec{w}_{[k]} \\ 0 \end{pmatrix} + \begin{pmatrix} A_{l_{[k]}} & \vec{b}_{l_{[k]}} \\ 0, \cdots 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{u}_{[k]} - 1_n \\ 0 \end{pmatrix} (t_{[k]} - t_{[k-1]}) \tag{5.11}$$

One may verify – using the properties of A_l and $\vec{b}_l$ from Section 4.3.1 – that $\begin{pmatrix} A_l & \vec{b}_l \\ 0, 0, \cdots 0 & 1 \end{pmatrix}$, $l \in \{0, 1, \cdots 2^{n-1}\}$ are idempotent and, thus, the above system represents the system (5.6) written in one dimension higher. Therefore, our system model written in equation (4.17) can be stabilized if condition (5.3) is satisfied. ■

The criterion in the above theorem may seem to be complex to apply since it requires 2^n matrix checks (see condition (5.3)). However, all the resulting $(n+1)2^n$ conditions are of the form:

$$\mu p_{i'} - \sum_{i \in S_{i'}} \alpha_{i'i} p_i - \beta_{i'} p_{n+1} \geq 0, \qquad \forall S_{i'} \subseteq \{1, 2, \cdots n\} \setminus \{i'\}, \quad \forall i' \in \{1, 2, \cdots n\}$$

By remembering that all coefficients are positive values, the above conditions all are satisfied if the following n conditions are satisfied:

$$\mu p_{i'} - \sum_{\substack{i=1 \\ i \neq i'}}^{n} \alpha_{i'i} p_i - \beta_{i'} p_{n+1} \geq 0, \qquad \forall i' \in 1, 2, \cdots n \tag{5.12}$$

Thus the criterion in Theorem 2 simplifies to only checking the above n conditions, where n is the number of resources in the system.

5.2.1 Sufficient Condition for a System of Two Resources

We shall now exemplify how to apply the above conditions on the example presented in Section 4.4, Figure 4.5. The matrices B_l are:

$$B_0 = \begin{pmatrix} \mu & 0 & -\beta_1 \\ 0 & \mu & -\beta_2 \\ 0 & 0 & \mu \end{pmatrix}, \qquad B_1 = \begin{pmatrix} \mu & 0 & 0 \\ -\alpha_{21} & \mu & -\beta_2 \\ 0 & 0 & \mu \end{pmatrix},$$
$$B_2 = \begin{pmatrix} \mu & -\alpha_{12} & -\beta_1 \\ 0 & \mu & 0 \\ 0 & 0 & \mu \end{pmatrix}, \qquad B_3 = \begin{pmatrix} \mu & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \mu \end{pmatrix}$$

Theorem 2 requires that there exists a coefficient $0 < \mu < 1$ and a point $[p_i]_3 \succ 0_3$ such that the following 12 conditions are satisfied:

$$B_1[p_i]_3 \succeq 0_3 \Rightarrow \begin{cases} \mu p_1 - \beta_1 p_3 \geq 0 \\ \mu p_2 - \beta_2 p_3 \geq 0 \\ \mu p_3 \geq 0 \end{cases}$$

$$B_2[p_i]_3 \succeq 0_3 \Rightarrow \begin{cases} \mu p_1 \geq 0 \\ \mu p_2 - \alpha_{21} p_1 - \beta_2 p_3 \geq 0 \\ \mu p_3 \geq 0 \end{cases}$$

$$B_3[p_i]_3 \succeq 0_3 \Rightarrow \begin{cases} \mu p_1 - \alpha_{12} p_2 - \beta_1 p_3 \geq 0 \\ \mu p_2 \geq 0 \\ \mu p_3 \geq 0 \end{cases}$$

$$B_1[p_i]_3 \succeq 0_3 \Rightarrow \begin{cases} \mu p_1 \geq 0 \\ \mu p_2 \geq 0 \\ \mu p_3 \geq 0 \end{cases}$$

When applying the sufficient conditions for this example, in order to determine $[p_i]_3 \succ 0_3$ we obtain the above system of 12 inequalities that further simplifies into the following system of only two inequalities (according to (5.12)):

$$\begin{cases} \mu p_2 - \alpha_{21} p_1 - \beta_2 p_3 \geq 0 \\ \mu p_1 - \alpha_{12} p_2 - \beta_1 p_3 \geq 0 \end{cases}$$

This system has the solutions:

$$p_1 \geq \frac{\beta_1 + \frac{\alpha_{12}\beta_2}{\mu}}{\mu - \frac{\alpha_{12}\alpha_{21}}{\mu}} p_3 \quad \text{and} \quad p_2 \geq \frac{\beta_2 + \frac{\alpha_{21}\beta_1}{\mu}}{\mu - \frac{\alpha_{12}\alpha_{21}}{\mu}} p_3$$
$$\text{if} \quad \mu - \frac{\alpha_{12}\alpha_{21}}{\mu} > 0 \text{ and } p_3 > 0$$

Ultimately, the example system from Figure 4.5 is stable if:

$$\alpha_{12}\alpha_{21} < \mu^2 < 1 \tag{5.13}$$

The meaning of condition (5.13) is that when accumulation of execution times moves circularly through the system e.g. from $\mathbf{N}_1$ to $\mathbf{N}_2$ and then back again to $\mathbf{N}_1$, the final accumulation on $\mathbf{N}_1$ is less then the initial one. For the particular case presented on page 65, we have $\alpha_{12} = \max\{\frac{3}{5}/\frac{1}{5}, \frac{3}{5}/\frac{3}{5}\} = 3$ and $\alpha_{21} = \max\{\frac{3}{5}/\frac{1}{5}, \frac{1}{5}/\frac{1}{5}\} = 3$. We then have $\alpha_{12}\alpha_{21} = 9 > 1$, thus the system is unstable, as previously illustrated following Table 5.1.

5.3 Stability Analysis

In this section we derive a condition on the behavior of the resource manager, that will render the whole adaptive real-time system, modeled in Section 4.2, stable.

Theorem 3:
For any system that satisfies conditions (5.1) and (5.3) (Theorems 1 and 2) and for any $\Omega \geq 2\frac{|\vec{w}^{\max}|_p}{1-\mu}$, a sufficient condition for stability is that its resource manager satisfies:

$$\vec{\rho}_{[k]} \in \begin{cases} \Gamma_\star, & \text{if } |\vec{x}_{[k]}|_p \geq \Omega \\ \mathbf{P}, & \text{otherwise} \end{cases}; \qquad \forall k \in \mathbb{Z}_+ \tag{5.14}$$

$\diamond$

PROOF We have shown in the above section that when choosing rate vectors from $\Gamma_\star$, a norm of the state of the system reduces below $2\frac{|\vec{w}^{\max}|_p}{1-\mu}$. Thus, while $\vec{x}_{[k]} \geq \Omega$ as the resource manager assigns rate vectors $\vec{\rho} \in \Gamma_\star$, the state norm will reduce until it becomes lower than Ω.

When the state of the system is such that $|\vec{x}_{[k]}|_p < \Omega$ the state norm may increase, but the increase is limited to the number of jobs that can enter the system in between two resource manager actuations. Assuming the highest task graph rates ($\vec{\rho}^{\max}$) the norm of the state may grow until it reaches the bound:

$$|\vec{x}_{[k]}|_p \leq \Psi = \Omega + \max_{i \in \mathcal{I}_{\mathfrak{R}}} \left\{ \frac{\sum\limits_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \lceil \rho_{\gamma(j)}^{\max} h \rceil}{p_i} \right\} \tag{5.15}$$

at witch point the state norm reduces again. Thus the system controlled by any resource manager that satisfies (5.14) is stable. ∎

The above theorem is not as straightforward to apply as it may appear because the resource manager does not actuate at every moment in time $t_{[k]}$ (see Section 4.1), yet condition (5.14) must be satisfied for all time moments. This has implications on how the resource manager can choose task chain rate vectors when it must do so from the set $\Gamma_\star$. When the resource manager actuates, the newly chosen rates take effect during several time moments following the completion of the resource manager routine. At these time moments, the rate vectors contain a mixture of new task chain rates, for some chains, and old task chain rates for the rest. The rate vector choice taken by the resource manager must be so that these intermediate rate vectors belong to $\Gamma_\star$ as well.

To understand this, let us consider a system with two task chains which evolves as in Figure 5.1. Let us consider the situation when the resource manager runs at the time moments: $t_{[k]}$, $t_{[k']}$, $t_{[k'']}$, etc. and starting with $t_{[k]}$ it must choose rates from $\Gamma_\star$. After finishing its execution at $t_{[k]} + \Delta$, the new jobs of the task chains will be released

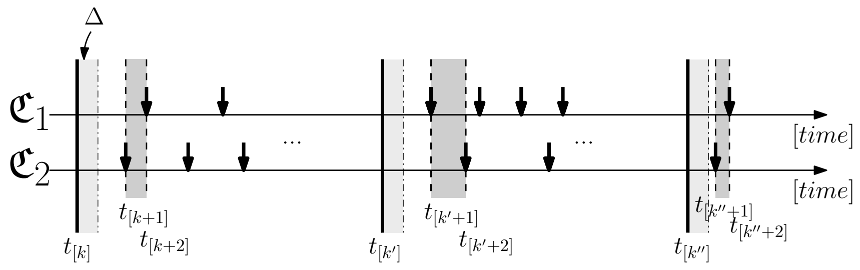


Figure 5.1: Behavior of a system with two tasks chains.

at their new task rates at time moments $t_{[k+1]}$ for $\mathfrak{C}_2$ and $t_{[k+2]}$ for $\mathfrak{C}_1$. Similarly will happen at $t_{[k']}$, $t_{[k'']}$, etc. and we observe that the rates selected at $t_{[k]}$ (denoted $\vec{\rho}_k \in \Gamma_\star$) take effect in the interval of time $[t_{[k+2]}, t_{[k'+1]}]$ and the rates selected at $t_{[k']}$ ($\vec{\rho}_{k'} \in \Gamma_\star$) take effect in the interval of time $[t_{[k'+2]}, t_{[k''+1]}]$ and so on. During, for example, the interval of time $[t_{[k'+1]}, t_{[k'+2]}]$, the system finds itself in a situation where the rate vector is $\vec{\rho}_{[k'+1]} = (\rho_{1(k')}, \rho_{2(k)})^T$ of which we do not know whether it belongs to $\Gamma_\star$ or not.

Resource managers must be designed such that when choosing rates from $\Gamma_\star$ they do so such that they can guarantee that all intervals of time $[t_{[k'+1]}, t_{[k'+2]}]$, $[t_{[k''+1]}, t_{[k''+2]}]$, etc. will have rate vectors from $\Gamma_\star$ as well. This can be achieved by requiring the resource manager to satisfy condition (5.14) only at all time moments $t_{[k']}$ when it actuates, and by adding extra conditions which guarantee that condition (5.14) is satisfied for the rest.

An example of such extra condition is:

$$\begin{aligned} \vec{\rho}_{[k']} \preceq \vec{\rho}_{[k]} \quad & \text{if } |\vec{x}_{[k]}|_p \geq \Omega \text{ and } |\vec{x}_{[k']}|_p \geq \Omega, \\ & \forall k, k' \in \mathbb{Z}_+ \text{two successive resource manager actuations} \end{aligned}$$

which requires that, while the manager must choose rate vectors from $\Gamma_\star$, it does so by choosing vectors with progressively smaller components. Since all intermediate rate vectors (at the intermediate time moments between $t_{[k]}$ and $t_{[k']}$, containing combinations of the new and old rates) have smaller/equal rates than $\vec{\rho}_{[k]} \in \Gamma_\star$, they are all part of $\Gamma_\star$ and, thus, condition (5.14) is satisfied for all time moments.

5.4 Geometric Interpretation

In this section we aim at presenting geometric interpretations of each of the three theorems developed in this chapter.

5.4.1 Geometric Interpretation of the Set $\Gamma_\star$

We present the geometric interpretation of the set $\Gamma_\star$ with help of the system in Figure 5.2. This is a system featuring two task chains and four resources. For this system the set of all possible rate vectors $\mathbf{P}$ is a subset of $\mathbb{R}^2_+$ and the set $\Gamma_\star \subset \mathbf{P}$ is bounded by the following constraints:

$$\vec{u}_1 = c_1^{\max}\rho_1 + c_6^{\max}\rho_2 \leq 1 \qquad \vec{u}_2 = c_2^{\max}\rho_1 \leq 1$$
$$\vec{u}_3 = c_5^{\max}\rho_2 \leq 1 \qquad \vec{u}_4 = c_3^{\max}\rho_1 + c_4^{\max}\rho_2 \leq 1$$

The sets $\mathbf{P}$ and $\Gamma_\star$ are illustrated in Figure 5.3 where the boundary figured in black represents all rate vectors that, assuming worst-case execution times, keep one or more resources $\mathbf{N}_i$, $i \in \mathbf{P}$ at load $\vec{u}_i = 1$, and thus, keep the system at the limit of its schedulability. Any other rate vectors from $\Gamma_\star$ (the gray surface) will keep all resources underloaded in the worst-case.

5.4.2 Geometric Interpretation of the Worst-Case Behavior of Adaptive Real-Time Systems

In this section we present the geometric interpretation that led us to build the proof of Theorem 2. Let us start first by analyzing our system's model for the worst-case from equation (4.17). Our model has the following features. It is:

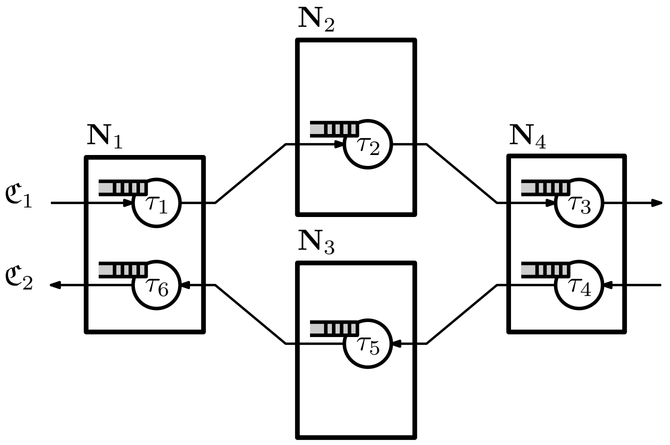


Figure 5.2: An example of a system with two task chains and four resources.

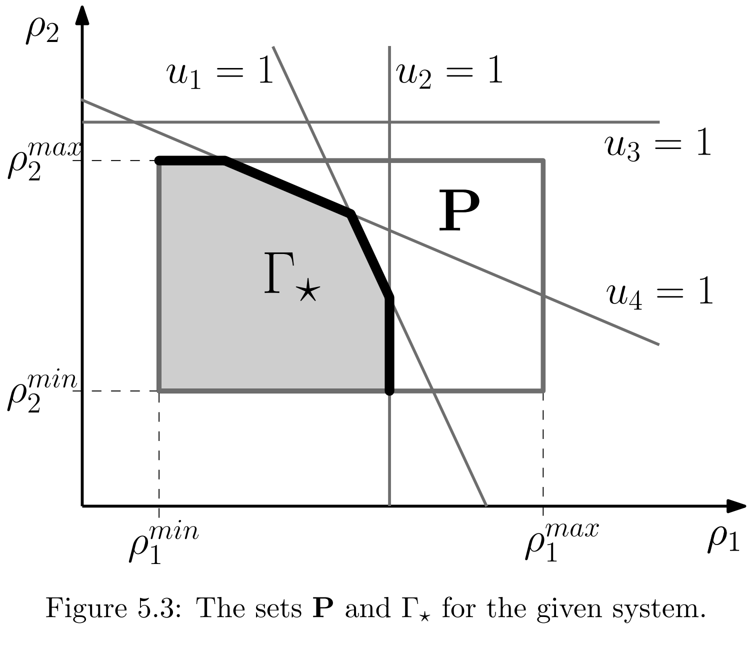


Figure 5.3: The sets $\mathbf{P}$ and $\Gamma_\star$ for the given system.

- *linear* – because the formula to determine $\vec{x}_{[k+1]}$ is a linear combination of $\vec{x}_{[k]}$ (the current state), $(\vec{u}_{[k]} - 1_n)(t_{[k+1]} - t_{[k]})$ (the external inputs), and $(\vec{w}_{[k+1]} - \vec{w}_{[k]})$ (the perturbations – also inputs),

- *non-autonomous* – because its inputs are not 0,

- *non-homogeneous* – because of the extra $\vec{b}_{l_{[k]}}$, and

- has *random switching* – because $A_{l_{[k]}}$ and $\vec{b}_{l_{[k]}}$ are changing with time.

We present our geometric interpretation on a system having the following simplifying assumptions, that help reduce its behavioral complexity and allows us to understand its basic underlying behavior:

- $\vec{u}_{[k]} = 1_n$ and $\vec{w}_{[k+1]} = \vec{w}_{[k]}$ which makes our system autonomous, and

- $\vec{b}_{l_{[k]}} = 0_n$, which makes our system homogeneous.

These simplifying assumptions do not greatly affect the behavior of our system. We can expect that the perturbations are approximatively 0_n since $\vec{w}_{[k]}$, for all k, is bounded ($0_n \preceq \vec{w}_{[k]} \preceq \vec{w}^{\max}$, $\forall k$). The choice of inputs satisfies at the limit the necessary conditions (see equation (4.20) and condition (5.1)), implying that the load on each resource in the system is 1. Choosing any other inputs from $\Gamma_\star$ allows for smaller loads and, thus, allows for "more stable" behaviors. Homogeneity does not affect the behavior of our system since any non-homogeneous system can be transformed into a homogeneous one by writing it in homogeneous coordinates, as done for step 3 in the proof of Theorem 2.

With these simplifying assumptions, our system becomes:

$$\vec{x}_{[k+1]} = A_{l_{[k]}}\vec{x}_{[k]} \text{ where } A_{l_{[k]}} \in \{A_0, A_1, \cdots, A_{2^n-1}\}$$

and its solution is:

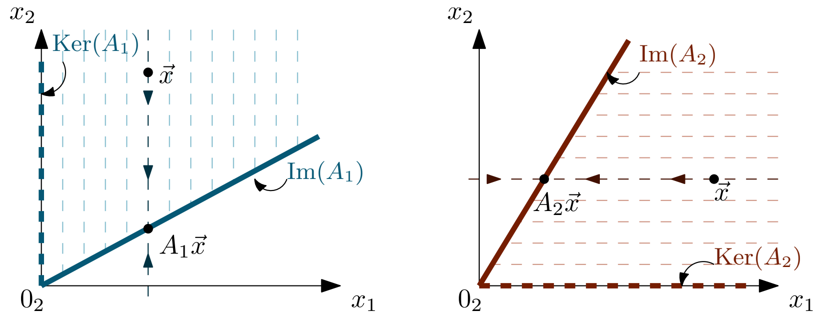
$$\vec{x}_{[k+1]} = A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[0]}} \vec{x}_{[0]}$$

The stability of the system depends on the stability of the *discrete linear inclusion* $A_{l_{[k]}} A_{l_{[k-1]}} \cdots A_{l_{[0]}}$. Such discrete linear inclusions are unstable in the case when at least one of the matrices A_l, $l \in \{0, 1, \cdots, 2^n - 1\}$ has at least one eigenvalue larger than 1 (see [Dau92, Dau01, Har02]) and they satisfy various stability properties when all eigenvalues, of all matrices, are strictly smaller than 1. For the systems that we model, however, all matrices are oblique projections and, thus, have eigenvalues which are 0 or 1 making our systems live in the borderline between stable and unstable. Nevertheless, a linear switching system is stable if there exists a norm $|\cdot|$ on the state of the system (see, for example [Gug05, Gug09, Liu13, Dai13, Tei12, Gru67]) such that:

$$|A_l \vec{x}| \leq |\vec{x}|, \quad \forall l \in \{0, 1, \cdots, 2^n - 1\}, \forall \vec{x} \in \mathcal{X}$$

We shall now explain the idea behind our norm $|\cdot|_p$ for the case of systems featuring 2 resources (of which the system from Figure 4.5 is one example). The trajectory of the state of such a system is bounded in the positive quadrant of a plane, whose origin point represents the desired state of the system, with 0 accumulation of execution time for both resources. Such a system has $2^2 = 4$ branches between which it can switch, each branch characterized by a matrix A_l. As said previously, all matrices are oblique projection matrices, which project a point $\vec{x}$ in space parallel with their kernel $\mathrm{Ker}(A_l)$ onto their image $\mathrm{Im}(A_l)$ (see Figure 5.4). For the $2D$ case we have the following 4 matrices together with the eigenvectors of their kernel and image spaces:

$$A_0 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \qquad \mathrm{Ker}(A_0) : \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}; \ \mathrm{Im}(A_0) : \left\{ \right\}$$



(a) image and kernel space of A_1 (b) image and kernel space of A_2

Figure 5.4: Geometric interpretation of the kernel and image spaces, and the projection effect of matrices A_1 and A_2.

$$A_1 = \begin{pmatrix} 1 & 0 \\ \alpha_{21} & 0 \end{pmatrix}, \quad \mathrm{Ker}(A_1) : \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}; \qquad \mathrm{Im}(A_1) : \left\{ \begin{pmatrix} 1 \\ \alpha_{21} \end{pmatrix} \right\}$$

$$A_2 = \begin{pmatrix} 0 & \alpha_{12} \\ 0 & 1 \end{pmatrix}, \quad \mathrm{Ker}(A_2) : \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}; \qquad \mathrm{Im}(A_2) : \left\{ \begin{pmatrix} \alpha_{12} \\ 1 \end{pmatrix} \right\}$$

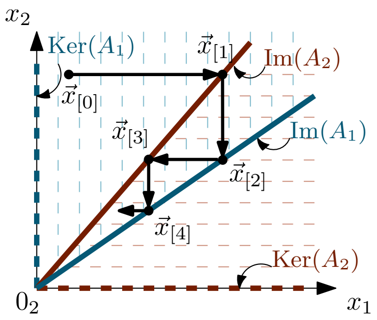
$$A_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathrm{Ker}(A_3) : \left\{ \right\}; \qquad \mathrm{Im}(A_3) : \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$

Matrices $A_0 = 0_{n\times n}$ and $A_3 = I_n$ are not interesting as any infinite matrix product that includes $0_{n\times n}$ is $0_{n\times n}$ and any product that includes I_n remains unchanged. Also, the idempotence property of oblique projection matrices ($A_1^2 = A_1$ and $A_2^2 = A_2$) means that for the case of systems with two resources, the only infinite matrix products that are of concern are the following four:

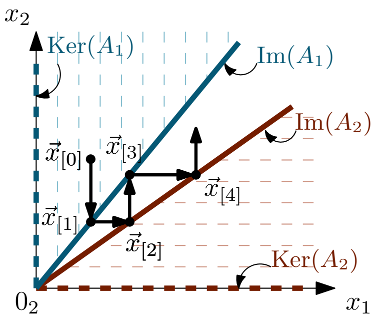
$$\vec{x}_{[k]} = A_2 A_1 A_2 A_1 \cdots A_1 \vec{x}_{[0]} = (\alpha_{12}\alpha_{21})^{\lfloor \frac{k}{2} \rfloor} \begin{pmatrix} 1 & 0 \\ \frac{1}{\alpha_{12}} & 0 \end{pmatrix} \vec{x}_{[0]}$$

$$\vec{x}_{[k]} = A_2 A_1 A_2 A_1 \cdots A_2 \vec{x}_{[0]} = (\alpha_{12}\alpha_{21})^{\lfloor \frac{k}{2} \rfloor} A_2 \vec{x}_{[0]}$$

$$\vec{x}_{[k]} = A_1 A_2 A_1 A_2 \cdots A_1 \vec{x}_{[0]} = (\alpha_{12}\alpha_{21})^{\lfloor \frac{k}{2} \rfloor} A_1 \vec{x}_{[0]}$$



(a) stable system



(b) unstable system

Figure 5.5: The evolution in time of two systems.

$$\vec{x}_{[k]} = A_1 A_2 A_1 A_2 \cdots A_2 \vec{x}_{[0]} = (\alpha_{12}\alpha_{21})^{\lfloor \frac{k}{2} \rfloor} \begin{pmatrix} 0 & \frac{1}{\alpha_{12}} \\ 0 & 1 \end{pmatrix} \vec{x}_{[0]}$$

In all cases, the state reduces if $\alpha_{12}\alpha_{21} < 1$, which is the result obtained in condition (5.13).

In Figures 5.4a and 5.4b we present the kernel and image space, of matrices A_1 and A_2 respectively, together with their projective effect when applied to a certain state of the system. For both matrices, we can observe that the projection is done parallel with their kernel spaces, and that during projection one of the dimensions of the state remains constant while the other decreases when the state lies in the non-negative cone formed by the kernel and image space, and increases when the state lies outside this cone.

In Figure 5.5 we present the evolution of two systems with two resources. For the stable system (Figure 5.5a) we can observe that, with the possible exception of the first projection ($\vec{x}_{[0]} \to \vec{x}_{[1]}$), the state of the system moves closer and closer to the origin. For the unstable system (Figure 5.5b), however, apart from the first projection, the state of the system moves farther and farther away from its desired stable state (0_2). The geometric difference between the two systems is that for the stable one, the image space of A_1 is included in the non-negative cone formed by the kernel and image space of

A_2 ($\text{Im}(A_1) \subset \angle(\text{Ker}(A_2)0_2\text{Im}(A_2))$), and vice-versa. For the unstable case, this condition does not hold. The significance of this observation is the following: during the evolution of the system, the state get projected, successively, from one image space to the next and, as we have seen from Figure 5.4, states within the non-negative cone reduce some of their dimensions when projected on the respective matrix. The states from outside the cone increase some of their dimensions, thus, successive projecting leads to increase in state and instability.

To express the idea that the non-negative cones formed by the kernel and image spaces of all matrices must overlap, we say that there must exist a point $\vec{p}$ in the state space, that belongs to all cones. Since our state space is the positive quadrant of an n-dimensional space (n being the number of resources in the system), the point $\vec{p}$ will have all its coordinates (written in the standard basis of the space $\mathbb{R}^n$) as positive values. If this point belongs to the non-negative cone formed by the kernel and image space of a matrix, then its coordinates, written in the basis formed by the eigenvectors of the kernel and image space of that matrix, should all be positive values. For our system of 2 resources, the basis associated with the 4 matrices and non-negative cones are:

$$A_0 : \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \qquad A_1 : \left\{ \begin{pmatrix} 1 \\ \alpha_{21} \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$

$$A_2 : \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \alpha_{12} \\ 1 \end{pmatrix} \right\} \qquad A_3 : \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$

and the above condition, written for every basis of the 4 matrices becomes:

$$A_0 : \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^{-1}}_{B_0} \vec{p} \quad \succeq 0_2, \quad A_1 : \underbrace{\begin{pmatrix} 1 & 0 \\ \alpha_{21} & 1 \end{pmatrix}^{-1}}_{B_1} \vec{p} \quad \succeq 0_2$$

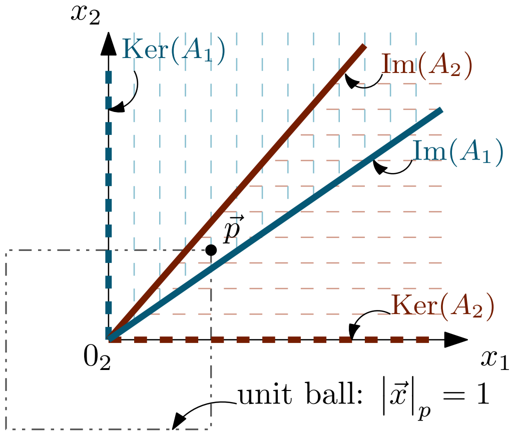


Figure 5.6: The unit ball associated with norm $|\cdot|_p$.

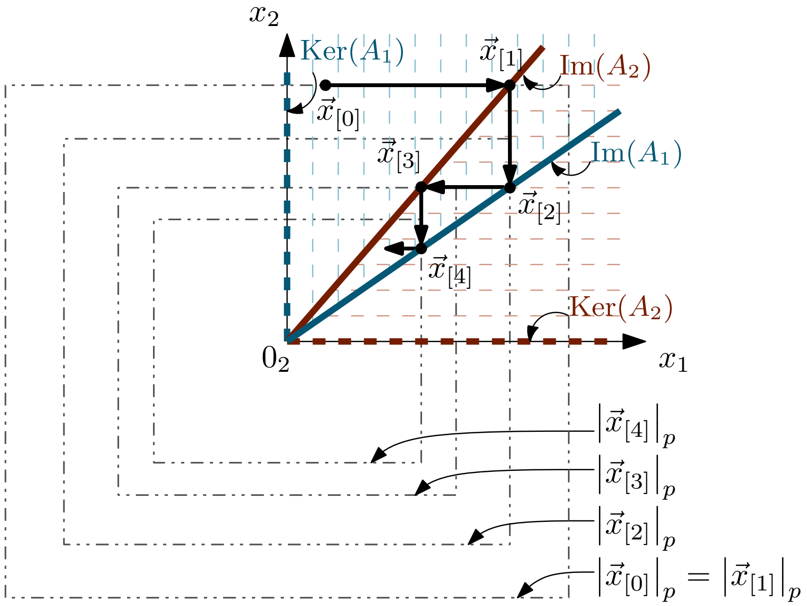


Figure 5.7: The evolution of a stable system, together with the norms of the figured states and their respective balls.

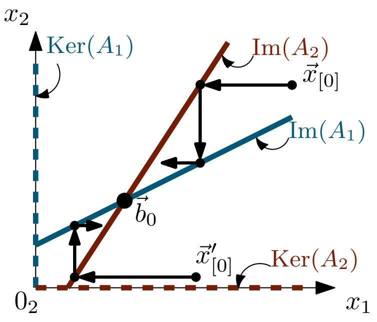


Figure 5.8: The evolution of a non-homogeneous stable system.

$$A_2 : \quad \underbrace{\begin{pmatrix} 1 & \alpha_{12} \\ 0 & 1 \end{pmatrix}^{-1}}_{B_2} \vec{p} \quad \succeq 0_2 \qquad A_3 : \quad \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^{-1}}_{B_3} \vec{p} \quad \succeq 0_2$$

These conditions are, in fact, condition (5.3) if $\mu = 1$. By choosing $\mu < 1$ we are effectively limiting the point $\vec{p}$ to strictly belong to the interior of each cone, which is needed for the general system from equation (4.17).

With the point $\vec{p}$ determined, we can build our norm $|\cdot|_p$ whose unit ball is illustrated in Figure 5.6. As we can observe, the unit ball is a polytope whose edges are parallel with the axis of the standard system of coordinates. Since the kernels of all of our matrices are spaces generated by combinations of the eigenvectors of the standard basis (along some of these axis), projections on any image space (along any kernel space) cannot increase the norm the state. Figure 5.7 shows how the norm of the state reduces with each successive projection.

Finally, let us discuss the effects of non-homogeneities, inputs and perturbations upon the system. Non-homogeneities change the stationary point of the system from 0_n to $\vec{b}_0 = (\beta_1, \beta_2)^T$, as shown in Figure 5.8 while the stable or unstable behavior remains unaffected. What this figure illustrates is the worst-case behavior of the system, as obtained after the approximation done via inequality (4.16). The

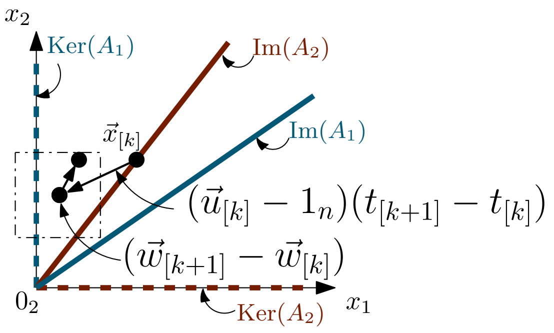


Figure 5.9: The evolution of a non-autonomous stable system

real worst-case behavior of the system will lie in between 0_n and $\vec{b}_0$ as the coefficients β, obtained out of inequality (4.16) are only an upper bound to the real behavior, which is variable.

In Figure 5.9 we present the effects of the perturbations and inputs into the system. Perturbations will move the current state to a location in its vicinity, before it gets projected onto one of the matrices. This movement may lead to increases in the norm of the state, however, as shown in the proof of Theorem 2, these increases are bounded and do not affect stability. The inputs of the system, if chosen from the set $\Gamma_\star$ such that $\vec{u}_{[k]} \prec 1_n$, will have the effect of bringing the state closer to 0_n, before projection, and thus, improve the rate at which the system converges.

5.4.3 Geometric Interpretation of the Stability Condition

We present the meaning of the stability condition (Theorem 3) with the help of Figure 5.10 where we illustrate a qualitative representation of the bounds Ω and Ψ, together with the decisions taken by the resource manager and the trajectory of the state. When the state of the system approaches 0_n the evolution becomes more unpredictable, due to the large effect that perturbations have onto the system. The

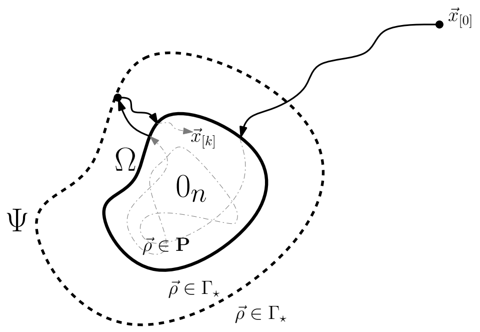

$\vec{x}_{[0]}$
$\vec{x}_{[k]}$
Ω
0_n
Ψ
$\vec{\rho} \in \mathbf{P}$
$\vec{\rho} \in \Gamma_\star$
$\vec{\rho} \in \Gamma_\star$

Figure 5.10: Qualitative illustration of the behavior of a stable system and, the meaning of the Ω and Ψ bounds.

bound Ω has to be large enough to cover all these effects. Of course, the system rarely exhibits its worst-case behavior, therefore, when the state is sufficiently small (below Ω) the resource manager is free to choose any rates ($\vec{\rho} \in \mathbf{P}$) it deems necessary to achieve its goals. The system's state above Ω is, however, an indication that its behavior nears worst-case behavior and strong action ($\vec{\rho} \in \Gamma_\star$) is need to keep the system stable. Because the growth of accumulation of execution time is finite for any finite intervals of time, even if the system starts exhibiting worst-case behavior when its state is just below Ω, the effect will still be a jump within Ψ.

At start time, the system state may be, theoretically, extremely large and outside Ψ, however, a stable system (that satisfies our theorems) is *ultimately bounded* by the bound Ψ, meaning that the state eventually becomes trapped in the ball of size Ψ around the state 0_n. However, this bound is not reached *uniformly* meaning that it might take an infinite amount of time before the state of the system reduces below Ψ. For the adaptive distributed real-time systems that we model in this work, this is not a problem, as computer systems start with empty queues and, thus, $\vec{x}_{[0]} = 0_n$. However, for systems that require uniformity, this can simply be obtained by restricting the choice of task chain rates when $\vec{x}_{[k]} \geq \Omega$ to any subset of $\Gamma_\star$ that does not allow for worst-case input load of 1 to any of the system's resources (any subset of $\Gamma_\star$ that does not include the boundary marked in black from Figure 5.3).

6 Discussion

In this chapter we further explain some implications of our results, how to apply them, and their limitations. We show how our system model can be extended in several different aspects and we end the chapter by determining the link between our stability results and more classical real-time results obtained via timing analysis.

6.1 Interpretation of the Results

In this work we have given a mathematical model (equation (4.5)) for the evolution in time of a system composed of a number of resources and a number of task chains distributed across them. These task chains release jobs at variable rates, decided by a resource manager (equation (4.6)). Because we only consider minimal assumptions regarding the behavior of the schedulers (see Chapter 3) on all resources, we model the evolution of the queues in an aggregated fashion (we describe how the task queues of tasks running on each resource evolve collectively).

With the model that we have developed, our goal is to describe its worst-case behavior and determine if the modeled system is stable in that case. Stability implies bounded task queues which means bounded real-time properties such as worst-case response times for tasks and end-to-end delays for task chains. In order to describe the worst-case behavior of the system, we bring forth notations such as the accumulation of execution times on each resource and we proceed to rewrite and build several upper-bound approximations of our model (inequations (4.7), (4.12), and (4.13)) until we find a worst-case behavior model that we can work with (inequation (4.17)).

With the worst-case behavior model we proceed to determine stability properties and we present three results: necessary conditions for stability (Theorem 1), sufficient conditions for stability (Theorem 2), and, finally, conditions on the resource manager controlling the system (Theorem 3).

The necessary conditions observe the individual behavior of each resource and determine if there exist rate vectors that can independently keep the accumulation of execution times on each resource bounded. The outcome is the set $\Gamma_\star$ of all these rate vectors. These conditions, however, are not sufficient because they do not take into account the interaction between the resources (interaction determined by the mapping of task chains to resources).

The sufficient conditions look at the interaction between the resources and determine if choosing actuation rates from the set $\Gamma_\star$ indeed guarantees the non-increase of the state of the system.

The necessary and the sufficient conditions in Theorems 1 and 2 respectively, refer to the properties of the system (resources and tasks) and indicate if it can be guaranteed that there exists any resource manager that keeps the system stable. If those conditions are satisfied, Theorem 3 formulates a condition which, if satisfied by the resource manager, guarantees that the system will remain stable.

To determine if a particular real-time system can be controlled in a stable way, we have to perform the checks from Theorems 1 and 2.

All of these checks are performed off-line.

There are n necessary conditions to check (one for each resource in the system) and each is linear in the number of tasks running on that particular resource (see equation (5.1)). This makes the complexity of checking the necessary conditions to be linear in the number of tasks in the system, and thus easy to perform computationally.

The sufficient conditions require the determination of the coefficients $p_1, \ldots, p_{n+1}$ and μ. The conditions are composed of n equations (see equation (5.12)) linear in p_i, $1 \le i \le n$, therefore, for some fixed p_{n+1} and μ, they can be checked using a simplex algorithm. The coefficient p_{n+1} may be chosen arbitrarily (1 would be a convenient value) and the coefficient μ may be determined using a binary search on top of the simplex algorithm.

Theorem 3 can be used in different ways:

1. *To determine if an existing resource manager is stable.* This is done by determining if a certain threshold Ω, on the norm of the state, exists.

2. *To help build custom, ad-hoc resource managers which are stable.* This is done by designing the resource manager around a preselected threshold Ω.

3. *To modify an existing resource manager to become stable.*

We shall give several simple examples of resource managers in Section 7.

6.2 Extensions of the System Model

Here we will extend the system model presented so far by describing and addressing a number of its limitations. These limitations are:

1. The software applications in the system are composed of task chains. We will here describe how our modeling and analysis can be extended to more general task graphs.

2. The resource manager's actuation method is by changing task chain (or task graph) rates. We will extend our model to include more general methods of actuation, such as changing the resources' speed or job dropping/admission.

3. The resource manager's period is fixed. We will extend the model to consider varying period.

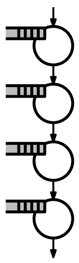
4. There is a gap between the necessary and the sufficient conditions a system must satisfy for stability. We shall present a discussion of where this gap comes from and we shall describe how to build algorithms that determine how to modify the system's parameters in order to determine stability for systems which do not satisfy the sufficient conditions.

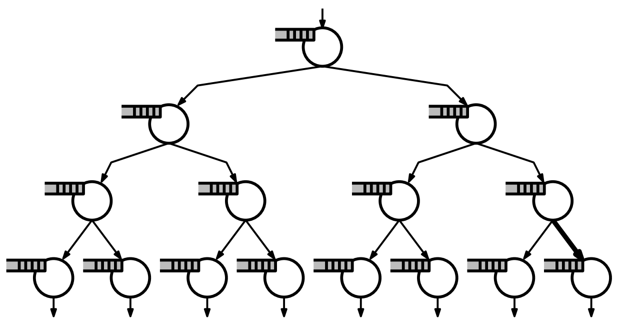
All the extensions that we will present in this section can be applied simultaneously.

6.2.1 Extensions to Task Graphs

We address here the limitation of our model being restricted to applications composed of task chains (Figure 6.1a: where every task in the chain has at most one direct predecessor and one direct successor) rather than the more general *acyclic task graphs*. We have presented our theoretical results considering this model, for simplicity. However, what we have used, in fact, is only the knowledge that each task has a single direct predecessor, useful to us in computing the quantities $\zeta_i^{\max}$ (see equation (4.10)). Thus, all results trivially extend to task trees (Figure 6.1b: each task has at most one direct predecessor but may have many direct successors) as well. Let us observe that in both cases, the task chain or task tree has only on task without predecessors, which is the *entry task* for newly released jobs in the system[1]. For any task in a task chain or task tree, there exists one and only one path of previous tasks that links it to the entry task.

[1]In this thesis we sometimes say that a job, when being released into the system, moves from task to task, from the beginning to the end of the task chain. In reality,


(a) task chain


(b) task tree

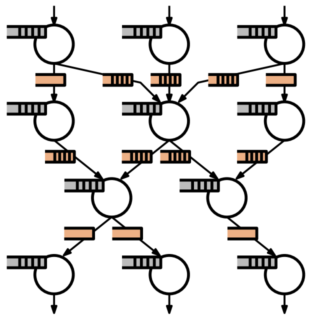

(c) task graph

Figure 6.1: Examples of software application structures.

Extending the model to general task graphs (Figure 6.1c: where each task may have several predecessors, as well) is more complex. In this case there exist several entry tasks to the task graph and multiple paths from a given task τ_j in the graph to several of these entry tasks. The total number of jobs accumulated on all these paths

jobs do not move from task to task. At release time, jobs of all tasks in the task chain are released simultaneously, but they are not ready for execution until their dependencies are satisfied. In a task chain, only one task would have its kth job ready for execution. All the predecessor tasks in the chain have already executed their kth job and all the successor task don't yet have their corresponding kth job ready for execution, therefore the impression that the kth job moves from task to task.

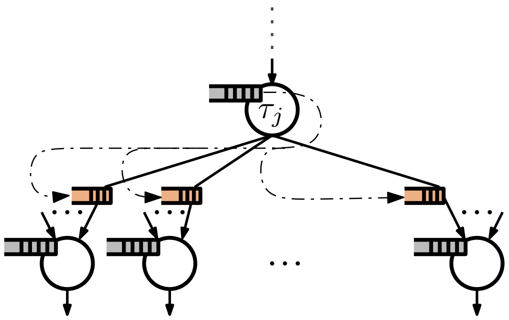


Figure 6.2: The completion of the kth job of τ_j sends a signal to all successors in the task graph to mark their kth job as ready for execution.

is the same and it corresponds to the number of jobs released, but not ready for execution of τ_j. Since jobs move faster on some paths than on others, we need to introduce the concept of *link queues*. Once the kth job of a task τ_j is executed (τ_{jk}), a signal is sent to all its successor tasks so that the corresponding kth jobs on the successor tasks can become ready for execution (see Figure 6.2). In the case of task chains and task trees that signal automatically sets the kth jobs of the successors as ready for execution, thus the jobs are marked in the task queues of their respective tasks. This happens because tasks have only one direct predecessor. In the case of general task graphs, however, a task τ_j may have multiple direct predecessors and the kth job has to be executed by each of these predecessors before it can become ready for execution for τ_j. If the kth job has been executed by one of the predecessors, the corresponding signal will wait in the link queue between it and τ_j until all other predecessors execute their kth job. Then τ_{jk} becomes ready to execute. In Figure 6.3 we show an example where task τ_j has three predecessors: $\tau_{j'}$, $\tau_{j''}$, and $\tau_{j'''}$. The link queues between $\tau_{j'}$ and τ_j, and $\tau_{j''}$ and τ_j have some signals inside them. No jobs can become ready for execution for τ_j because the queue between $\tau_{j'''}$ and τ_j is empty. As soon as a job of task

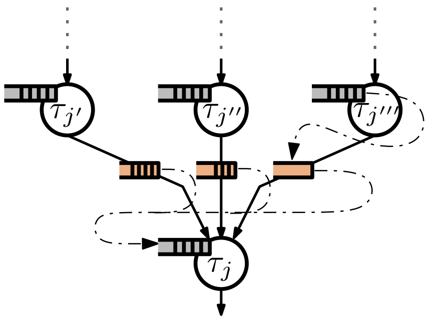


Figure 6.3: τ_j has three predecessors but the number of jobs that become ready for execution depends on $\tau_{j'''}$ as the link queue between it and τ_j is empty (this situation is valid as long as no other link queue becomes empty).

$\tau_{j'''}$ finishes, the signal of its completion is sent via the link into the link queue. At this moment the first element of each link queue is popped and a new job of τ_j becomes ready for execution. Thus, at each moment in time at least one of these link queues is empty.

Let us introduce the notation $_lq_j^{j'}$ to denote the link queue size associated with the dependency link from task $\tau_{j'}$ to task τ_j. Also, due to the fact that in task graphs tasks can have multiple direct predecessors we define the mapping:

- $\mathfrak{p}' : \mathcal{I}_\Theta \rightarrow \mathcal{P}(\mathcal{I}_\Theta)$, $\mathfrak{p}'(j)$ pmm' is the set of indexes of the direct predecessors of τ_j. $\mathfrak{p}'(j) = \emptyset$ if τ_j does not have predecessors.

We shall now rewrite the basic model of the system (equations (4.2), (4.3), and (4.4)) for the case of task graphs. The task queues for tasks in task graphs evolve in the same way as for task chains (see Section 4.2):

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k+1]} = \max\left\{ 0, \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} q_{j[k]} + \sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} s_{j[k+1]} - (t_{[k+1]} - t_{[k]}) \right\} \tag{6.1}$$

We now model the behavior of link queues. The jobs existing in the link queue ${}_lq^{j'}_{j[k+1]}$ evolve depending on the number of jobs executed by $\tau_{j'}$ during $[t_{[k]}, t_{[k+1]}]$: $\lceil q_{j'[k]} \rceil + s_{j'[k+1]} - \lceil q_{j'[k+1]} \rceil$; the number of jobs that become ready for execution for τ_j during the same interval: $s_{i[k+1]}$; and the number of jobs already existing in ${}_lq^{j'}_{j}$ at $t_{[k]}$: ${}_lq^{j'}_{j[k]}$.

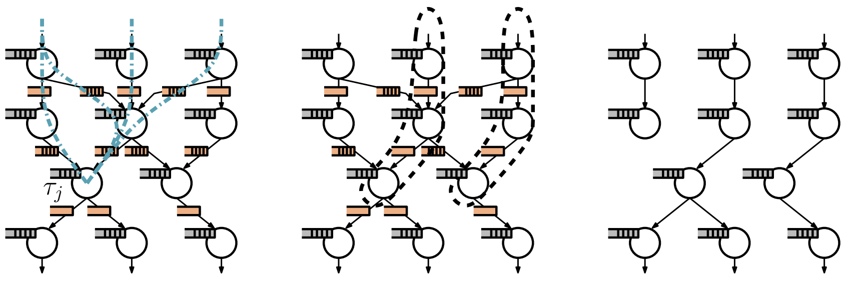
$${}_lq^{j'}_{j[k+1]} = {}_lq^{j'}_{j[k]} + \lceil q_{j'[k]} \rceil + s_{j'[k+1]} - \lceil q_{j'[k+1]} \rceil - s_{j[k+1]} \tag{6.2}$$

If a task has at least one predecessor, the number of jobs that become ready for execution during $[t_{[k]}, t_{[k+1]}]$ is given by the smallest number of jobs arriving on its incoming links. Otherwise, the number of jobs ready for execution depends on the rate of the task graph and is computed as for the task chain case:

$$s_{j[k+1]} = \begin{cases} \min\limits_{j' \in \mathfrak{p}'(j)} \{ {}_lq^{j'}_{j[k]} \lceil q_{j'[k]} \rceil + s_{j'[k+1]} - \lceil q_{j'[k+1]} \rceil \}, & \text{if } \mathfrak{p}'(j) \neq \emptyset \\ \lceil \rho_{\gamma(j)[k]} \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{\gamma(j)[k]}\} \rceil, & \text{otherwise} \end{cases} \tag{6.3}$$

The offsets of the system evolve in the same way as for task chains (see Section 4.2):

$$\phi_{p[k+1]} = \phi_{p[k]} + \frac{1}{\rho_{p[k]}} \lceil \rho_{p[k]} \max\{0, (t_{[k+1]} - t_{[k]}) - \phi_{p[k]}\} \rceil - (t_{[k+1]} - t_{[k]}) \tag{6.4}$$



(a) Possible paths for task τ_j (b) Critical paths in the task graph. (c) The series of task trees.

Figure 6.4: A task graph example and its behavior as a series of task trees.

The system model, in the case of applications formed of task graphs, is more complex than for systems with task chains as we have to model the link queues as well.

For the following discussion we define as *path switch (PS) interval* the time interval delimited by two events in which a link queue becomes empty. The system model can be reduced to that for task trees by observing that, for any given PS interval of time, for each task there exists (at least) one path from the task to an entry, such that along that path all link queues are empty. We call this path the *critical path* corresponding to the tasks[2]. The number of jobs that become ready for execution depends on the behavior of the tasks on the critical path. As long as the critical paths (for all tasks in the task graph) remain unchanged, we can model our task graph as a set of task trees, by removing the non-critical links (the links with non-empty link queues). In Figure 6.4 we show an example task graph where we can observe that for task τ_j we have multiple paths from τ_j to the entries (Figure 6.4a). For a given PS interval of time only one

[2]If there are several paths, for a given task, along which the queue links are empty, then the critical path can be chosen as being any of them, since they all behave in the same way.

of these paths is the critical path. In Figure 6.4b we show, during one PS interval, the critical paths for all tasks with multiple predecessors. This task graph, then behaves as if it was the series of task trees from Figure 6.4c. In this task graph example we have two tasks with 2 predecessors each, and one task with 3 predecessors. For any interval of time, therefore, the task graph will behave as one of $2 \cdot 2 \cdot 3 = 12$ different sets of task trees. Expanding our view to all task graphs in the system and considering the systems formed of all combinations of all sets of task trees possible, we can say that, at any moment of time, our system behaves as one of these systems of task trees. As time evolves, it switches between these systems in an unknown way. The number of systems of task trees depends on the number of predecessors that each task in the system has and is:

$$no\_of\_systems = \prod_{\substack{j \in \mathcal{I}_{\Theta} \\ \mathfrak{p}'(j) \neq \emptyset}} size(\mathfrak{p}'(j)),$$

where $size(\mathcal{S})$ is the cardinality of a set $\mathcal{S}$. For each system of task trees, we can build the coefficients α and β and the set of matrices A and vectors $\vec{b}$. The worst-case behavior model of our system of task graphs becomes a linear switching system, where sets the matrices A and of vectors $\vec{b}$ are the union of the corresponding sets of all the systems of task trees.

The analysis results for systems of task graphs remains the same as for task trees and task chains. But the complexity related to the satisfaction of Theorem 2 increases, as the number of matrices A and vectors $\vec{b}$ increases exponentially. Satisfying Theorem 2 implies satisfying equations (5.12), for a given μ and $[p_i]_{n+1}$, for all systems of task trees generated from the original system of task graphs. All these systems of task trees have different values for the coefficients α and β.

Looking at equations (5.12) we can observe that if they are satisfied for the system with the maximal values of the coefficients α and β, they are implicitly satisfied for all other systems as well. The max-

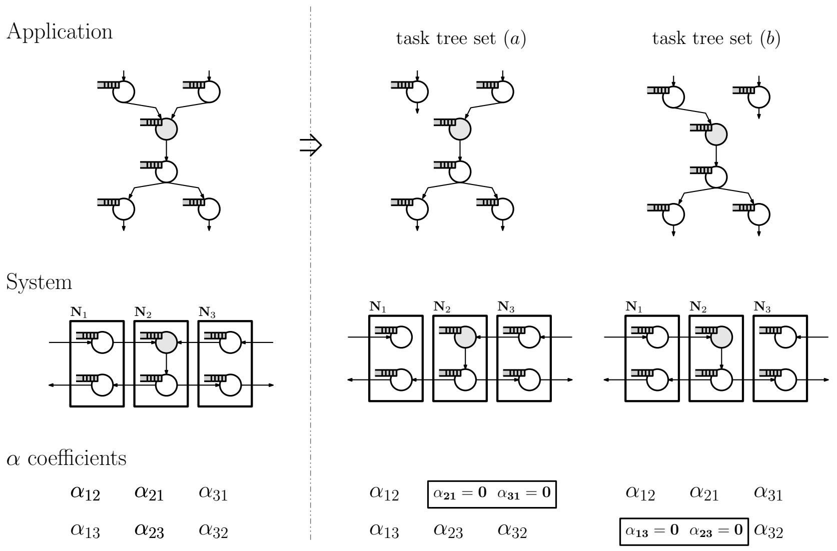


Figure 6.5: An example system containing a task graph, which behaves as one of two possible systems of task trees. None of the two systems have all α coefficients at their maximal value.

imal values of α and β can be obtained by simply not removing any of the links in the system (see equations (4.14) and (4.15)). *Solving the system of equations* (5.12) *for the coefficients α and β computed without removing any of the links from the task graphs can tell us whether the system can be stabilized.*

This condition is, however, pessimistic since it might happen that not all coefficients can have the maximal value at the same time. This is exemplified in Figure 6.5 where the system can behave like on of two possible systems of task trees: (a) and (b). This is because the system system has 6 tasks, one of the tasks has 2 dependencies, the rest having one or no dependency. All maximal values of the coefficients α, as calculated for the original system, are non-zero because the accumulation of execution times of all three resources are linked among themselves due to the data dependencies in the task graph.

However, we can observe that in the system (a), there is no link from resource $\mathbf{N}_1$ to the other resources, thus the coefficient α_{21} and α_{31}, in this system, are null. On the other hand, in the system (b) there is no link from resource $\mathbf{N}_3$ to the other two resources and, thus, the coefficients α_{23} and α_{13}, in this system, are null. Thus, while the system of task graphs has all coefficients non-zero, during its functioning it always behaves as if some of the coefficients are zero and, therefore, determining stability assuming all coefficients non-zero is pessimistic.

6.2.2 Extensions of the Resource Manager's Actuation Method

Let us now move on to the second type of limitations. In this paper we have used task chain rate changes as the method for adaptation of the system to various loads. This is useful e.g. for systems where the task chains represent controllers for external plants which may run with different control rates and higher rates improve the control quality. We shall discuss here extending the model for two other types of adaptations:

- adaptations by changing the resources' speed (using e.g. dynamic voltage and frequency scaling techniques) useful in low power systems and
- adaptations by job dropping/admission (useful e.g. in web and multimedia applications [Abd03]).

When dealing with adaptations by changing the resources' speed, let us assume constant task chain rates ρ_p, $p \in \mathcal{I}_{\mathfrak{A}}$. We consider the worst-case execution times of tasks computed at the slowest speed of each resource and we will model the act of increasing the speed of a resource as an increase in the load it can hold (as opposed to a decrease in the execution times of tasks running on that resource), e.g. doubling the speed of $\mathbf{N}_i$ means that $\mathbf{N}_i$ will handle a load of:

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_{j[k]} \rho_{\gamma(j)[k]} \leq 2$$

We then have, for each resource, a set of loads L_i associated with the speeds it can run at and the system model from (4.17) becomes:

$$\vec{x}_{[k+1]} \leq A_{l_{[k]}}(\vec{x}_{[k]} + (\vec{w}_{[k+1]} - \vec{w}_{[k]}) + (\vec{u}_{[k]} - [L_{i[k]}]_n)(t_{[k+1]} - t_{[k]})) + \vec{b}_{l_{[k]}} \tag{6.5}$$

The vector $[L_{i[k]}]_n$ is the input of this system, while $\vec{u}_{[k]}$ becomes a constant. In this setting, the necessary conditions (Theorem 1) describe the set of vectors $[L_{i[k]}]_n$ (the set of resource speeds) that can handle the worst-case load in the system ($\Gamma_\star^{speed}$). The sufficient conditions (Theorem 2) and the conditions on the resource manager (Theorem 3) retain their meaning and their proofs after the obvious modifications due to the different method of actuation. In the proof of the sufficient conditions, use the fact that $\vec{u}_{[k]} - [L_{i[k]}]_n \leq 0_n$ when choosing $[L_{i[k]}]_n$ from $\Gamma_\star^{speed}$ instead of $\vec{u}_{[k]} \leq 1_n$ when choosing rates from $\Gamma_\star$.

When dealing with job dropping/admission we consider again constant rates. The inputs in the system are the number of jobs dropped from each task[3] and represent the amount of load that is dropped from each resource $d_{i[k]}$. The necessary conditions then become

$$\sum_{j \in \mathcal{I}_{\mathbf{N}_i}} c_j^{\max} \rho_{\nu(j)} - 1 \leq d_i$$

and describe all combinations of dropped jobs that must occur in order to keep the system stable, in the worst-case scenario. In between two moments in time, $t_{[k]}$ and $t_{[k+1]}$, the quantity $d_i(t_{[k+1]} - t_{[k]})$ describes the accumulation of execution times that must be dropped on resource $\mathbf{N}_i$. Of course, this dropped accumulation may come as jobs of only

[3]when a job of a task τ_j is dropped, also the corresponding jobs from all the tasks that succeed τ_j in the task graph, are dropped.

one task and the system must allow for that. With our new inputs d_i the worst-case behavior of the system becomes:

$$\vec{x}_{[k+1]} \leq A_{l_{[k]}}(\vec{x}_{[k]}+(\vec{w}_{[k+1]}-\vec{w}_{[k]})+(\vec{u}_{[k]}-[d_{i[k]}]_n-1_n)(t_{[k+1]}-t_{[k]}))+\vec{b}_{l_{[k]}} \tag{6.6}$$

As before, the sufficient conditions and the conditions on the resource manager retain their meanings and proofs.

Indeed, all three actuation methods, that we have described so far, could be used simultaneously, without affecting the basic structure of the model and of the results.

6.2.3 Lifting the Restriction Regarding the Resource Manager's Period

In this thesis we have described and developed a theory for adaptive systems where the resource manager has a fixed actuation period h. We have done so for convenience, in order to reduce the number of definitions of parameters. For modeling and analysis purposes, however, we have not used this period at all, as we have described the evolution of our system at certain time instances which are not equally spaced and which are not linked with the resource manager period at all. We, therefore, can extend our model to include systems with resource managers that have a time varying actuation period.

From our assumptions, the actuation period must be larger than the time needed for the resource manager algorithm to finish $h_{[k]} \geq \Delta$, $\forall k$. Also, actuation needs to happen in order to control the state of the system, therefore $h_{[k]} \leq h^{\max} < \infty$, $\forall k$. For our system, therefore, any bounded variation in actuation period is allowable without modifying the underlying modeling and stability analysis presented so far. A minor modification appears in equation (5.15) though, where we must use $h^{\max}$ instead of h. In the following chapter we will give examples of how to compute several bounds on the response times of tasks in the system where we will make use of the period of the controller. Those bounds also need the same modification as above for the case of varying resource manager period.

6.2.4 Bridging the Gap between the Necessary and the Sufficient Conditions

In this section we address the limitations of the sufficient conditions. The sufficient conditions for stability (Theorem 2) determined in Section 5.2 test if a given system can be kept stable when applying task rate vectors from the set $\Gamma_\star$, determined in Section 5.1. If these conditions fail, the system may still be stable when rate vectors from only a subset of $\Gamma_\star$ are applied.

We shall introduce this idea with the help of two examples systems. Both examples have the topology of the system in Figure 4.5, both have the same minimum and maximum rate for their task chain, and the worst-case execution times for their tasks, and their sets $\Gamma_\star$ are given in Table 6.1.

For their given $\Gamma_\star$ ($\Gamma_\star^1$ for Example 1 and $\Gamma_\star^2$ for Example 2), both example satisfy condition (5.1) in Theorem 1. Because the worst-case execution times of Example 1 are larger or equal with the corresponding ones of Example 2, we have that $\Gamma_\star^1 \subset \Gamma_\star^2$.

Regarding the sufficient conditions, the system in *Example* 1 has:

$$\alpha_{12}^1 = \max\left\{\frac{4.95}{4.95}, \frac{4.95}{4.95}\right\} = 1 \qquad \alpha_{21}^2 = \max\left\{\frac{4.95}{5}, \frac{4.95}{5}\right\} = 0.99$$

thus $\alpha_{12}^1 \alpha_{21}^1 < 1$, satisfying condition (5.13). The system in *Example* 2, however, has:

$$\alpha_{12}^2 = \max\left\{\frac{4.95}{2}, \frac{4.95}{4.95}\right\} = 2.475 \quad \alpha_{21}^2 = \max\left\{\frac{4.95}{2}, \frac{2}{2}\right\} = 2.475$$

and it does not satisfy the sufficient conditions (5.13). It follows that the Example 1 presents a system that can be stabilized when selecting rates from $\Gamma_\star^1$, while Example 2 shows a system where selecting rates from $\Gamma_\star^2$ does not guarantee stability.

For the two example systems, as observed above, the worst-case execution times in Example 2 are smaller than the corresponding worst-case execution times in Example 1. Since Example 1 can be stabilized – even during its worst-case behavior, when the execution

Table 6.1: The parameters of the systems in Examples 1 and 2.

Example	$c_1^{\max}$	$c_2^{\max}$	$c_3^{\max}$	$c_4^{\max}$	$\Gamma_\star$
1	5	4.95	4.95	4.95	$\Gamma_\star^1 = [\rho^{\min}, 1/9.95] \neq \emptyset$
2	2	4.95	2	4.95	$\Gamma_\star^2 = [\rho^{\min}, 1/6.95] \neq \emptyset$

times are maximal – using rates from $\Gamma_\star^1$, it can also be stabilized using rates from the same set for better behaviors and, in particular, for the case when the execution times of jobs mimic the worst-case execution times from Example 2. This clearly suggests that, while Example 2 cannot be stabilized when selecting rates from $\Gamma_\star^2$, it nevertheless can be stabilized when selecting rates from $\Gamma_\star^1 \subset \Gamma_\star^2$.

In these examples, we assumed that $\rho^{\min}$ is a small enough value such that both sets, $\Gamma_\star^1$ and $\Gamma_\star^2$, are not empty. If we were to set $\rho^{\min} > 1/9.95$ we would find that there is no rate set that can stabilize the system in Example 2.

Returning to a general discussion, the question remains how to find a subset of $\Gamma_\star$ (of a given system) that keeps the system stable (if such a subset exists). Our sufficient conditions are related to the coefficients of the matrices B_l which are, as shown in Section 4.3.3, just ratios between the worst case execution times of various tasks in the system. We can alter the above coefficients by artificially considering in the analysis larger worst-case execution times for some of the tasks in the system. As long as the necessary conditions are still satisfied we obtain a new set $\Gamma'_\star \subset \Gamma_\star$. If the system with the altered coefficients satisfies the sufficient conditions, than the system is stable if $\Gamma'_\star$ is used instead of $\Gamma_\star$.

It is beyond the scope of this thesis to provide specific algorithms to manipulate systems in this way, in order to determine if a system is stable still can be kept stable, even if, for the initial $\Gamma_\star$, Theorem 2 is not satisfied.

6.3 Link with Timing Analysis

As mentioned previously, an upper bound on the norm of the state of the system determines upper bounds on the queue sizes of each task and maximum queue sizes determine worst-case response times of each task. We can, thus, see our work as a form of timing analysis for adaptive distributed real-time systems.

To determine upper bounds on the worst-case response times for the tasks in our system one may proceed as follows: it is common for computer systems to start with task queues of size 0, thus making the ultimate bound Ψ (equation (5.15)) a bound on the largest norm of the state of the system. Remembering the definition of our norm (equation (5.7)) and the definitions of the state and accumulation of execution time (Section 4.2), we may bound each task queue size in the system to $q_j \leq \left\lceil \Psi \frac{p_{\nu(j)}}{c_j^{\max}} \right\rceil$, $\forall j \in \mathcal{I}_{\mathbf{\Theta}}$, and assuming that all the jobs in the queue were released at their slowest rate, we see that the worst-case response time of task τ_j is

$$wcrt(\tau_j) = \frac{1}{\rho_{\gamma(j)}^{\min}} \left\lceil \Psi \frac{p_{\nu(j)}}{c_j^{\max}} \right\rceil + \max_{j' \in \pi(j)} \{wcrt(\tau_{j'})\}$$

In the above formula we use the knowledge that once the state of the system reaches its maximum $|\vec{x}_{[k]}|_p$, jobs must exit the system at the rate at which new jobs enter it, because the state may not grow anymore. Considering that the largest state is simply formed of jobs sitting in the queue of task τ_j, the slowest rate at which jobs – that will eventually reach τ_j – are released into the system and the size of the queue, give us the largest time that a job can take to clear the queue and move on. Knowing the worst-case response time of the previous task in the task chain, allows us to determine the worst-case response time of task τ_j, for any $j \in \mathcal{I}_{\mathbf{\Theta}}$. The analysis produced here is very general and, thus, potentially pessimistic. In Section 7.2.3 we present a different approach, for the particular case of uniprocessor systems and EDF scheduling.

7 Applications

IN THIS chapter we will show how to apply our stability criterion for several resource managers. In section 7.1 we discuss several examples of distributed systems, employing simple resource managers, and we show how to apply the three theorems described in this work. In section 7.2 we focus on uniprocessor systems and we discuss the stability of resource managers, performance aspects of different managers and how to compute real-time metrics such as worst-case response times for adaptive stable systems.

The framework for stability that we have presented so far is quite simple to apply. The steps needed to prove stability of an adaptive distributed real-time system are the following:

1. Start by having knowledge of th worst-case execution times of all tasks ($c_j^{\max}$, $\forall j \in \mathcal{I}_{\mathbf{\Theta}}$).
2. Verify conditions (5.1) (Theorem 1), and build the set $\Gamma_\star$ (equation (5.2)).
3. Compute the coefficients α and β (equations (4.14) and (4.15)).

4. Solve the system of equations (5.12) (Theorem 2) and determine the coefficient μ and vector $[p]_n$, if they exist.
5. If $[p]_n$ does not exist, try the ideas presented in Section 6.2.4, otherwise, move to the next step.
6. Verify condition (5.14) (Theorem 3) by doing the following:
   (a) Determine if the resource manager can select rate vectors (or, for the general case, input vectors) from $\Gamma_\star$.
   (b) Compute Ω by determining the largest state of the system, before the resource manager will select inputs from $\Gamma_\star$.

As we can see from the above 6 steps, determining stability mainly requires the verification of some simple linear constraints. We do not need to build exponentially many A_l and $\vec{b_l}$ matrices and vectors, and we do not need to understand the theories behind stability of switching systems.

7.1 Stability of Distributed Systems

In this section we present several example systems in order to further illustrate the meaning of our theorems and how to apply them in practice. We shall present simulation results for 7 example systems, all of which have the topology of the example presented in Section 4.4 (see Figure 7.1, which is identical to Figure 4.5) but with different system parameters (execution times, and rates) and different resource managers.

The first 4 examples employ a resource manager which always keeps the rates at their minimum. This allows us to explore different behaviors associated with satisfying or not satisfying the necessary and the sufficient conditions for stability. For these examples, at simulation, we only consider their worst case behavior (when all jobs execute with their worst-case execution times) and we plot the evolution in time of the task queue sizes.

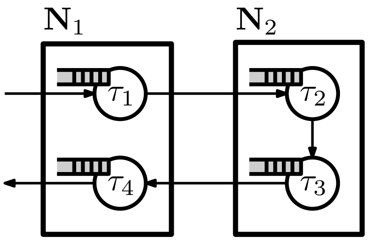


Figure 7.1: Example of a system with two resources and one task chain.

The last 3 examples present the case of a systems which satisfies both Theorems 1 and 2, running with different resource managers. We analyze for stability the resource managers in these examples and we plot the evolution in time of the queue sizes of each task and the utilization on the resources.

The parameters of interest for the first 4 examples (the worst-case execution times of tasks and the minimum rate of the task chain) are presented in Table 7.1.

Example 1 presents a system that cannot be stabilized because it does not satisfy the necessary conditions. We can observe that by performing the tests in Theorem 1: $\rho^{\min}(c_1^{\max} + c_4^{\max}) = 1.2 > 1$ for $\mathbf{N}_1$ and $\rho^{\min}(c_2^{\max} + c_3^{\max}) = 1.2 > 1$. The behavior of this system is depicted in Figure 7.2a where we observe the unbounded growth of q_1, thus showing that the system is unstable.

Example 2 presents a system which satisfies the necessary conditions from Theorem 1 but not the sufficient ones in Theorem 2. This is the example presented on page 65. Here we can see that $\alpha_{21} = \max\{\frac{c_2^{\max}}{c_1^{\max}}, \frac{c_3^{\max}}{c_1^{\max}}\} = 3$ and $\alpha_{21} = \max\{\frac{c_4^{\max}}{c_2^{\max}}, \frac{c_4^{\max}}{c_3^{\max}}\} = 3$ so that $\alpha_{12}\alpha_{21} = 9 > 1$, thus not satisfying the sufficient conditions from Theorem 2. The behavior of this system is presented in Figure 7.2b and we can observe the queue sizes increasing and decreasing in an oscillatory fashion. However, each successive increase is larger than the previous one, thus showing that the system spirals out of control.

Table 7.1: The parameters of the systems in Examples 1 to 4

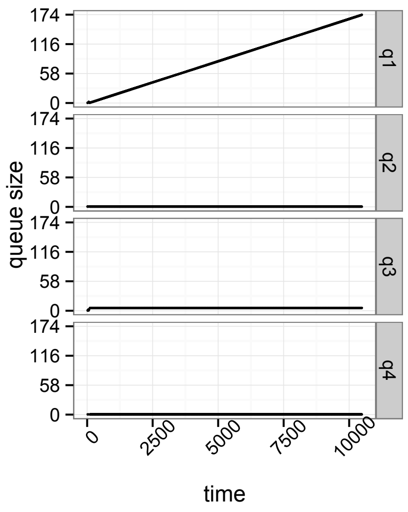
Example	$c_1^{\max}$	$c_2^{\max}$	$c_3^{\max}$	$c_4^{\max}$	$\rho^{\min}$
1	6	6	6	6	1/10
2	2	6	2	6	1/10
3	5	4	4	3	1/10
4	2	4.95	2	4.95	1/10

Example 3 presents a system that satisfies both the necessary and the sufficient conditions. This system can be stabilized by, for instance, always choosing the minimum rate. Its evolution is presented in Figure 7.2c and we can observe that the queue sizes are bounded for all queues in the system.

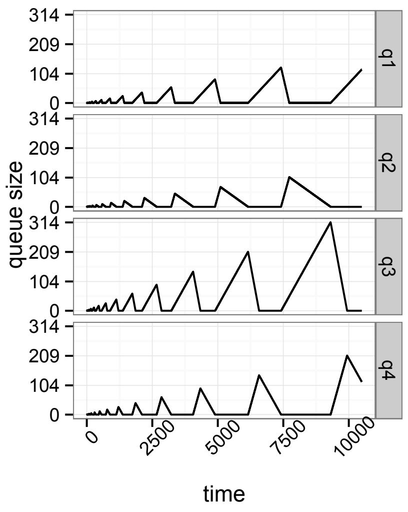
Example 4 presents a more interesting system, which shows an oscillatory but stable behavior, although while satisfying Theorem 1, it does not satisfy Theorem 2. This shows the sufficient, but not necessary nature of Theorem 2.

For the next three examples we present several simple resource managers and we show how to analyze them. All these resource managers control a system as the one in Figure 7.1 with the following parameters: $c_1 \in [0.5, 5]$, $c_2 \in [0.4, 4]$, $c_3 \in [0.4, 4]$, $c_4 \in [0.3, 3]$, and $\rho \in [0.1, 1]$. The resource managers actuate with a period $h = 100$ time units. This system satisfies Theorem 1, with $\Gamma_\star = \{\rho | \rho \in [0.1, 0.125]\}$, and Theorem 2, with $\mu = 0.882$ and $(p_1, p_2, p_3)^T = (100, 100, 1)$. Thus, it is possible to design resource managers that keep this system stable. In these examples we assume that high utilization on resources and low values for the queue sizes of the system are a desirable feature for our system. We shall first present the system in a setting where the task chain rates remain constant at a reference value ρ^{ref} and then we shall improve its performance by presenting the same system but featuring the two resource managers presented in Algorithms 1 and 2.

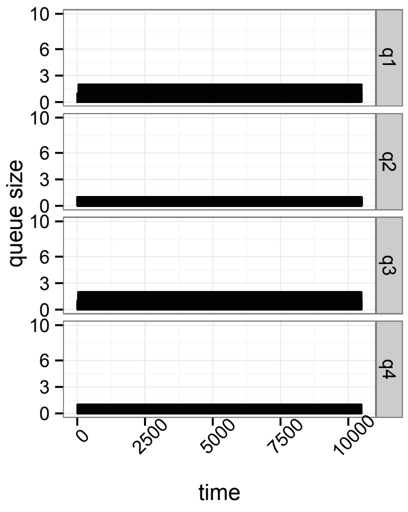
The resource manger presented in Algorithm 1 is a simple pro-



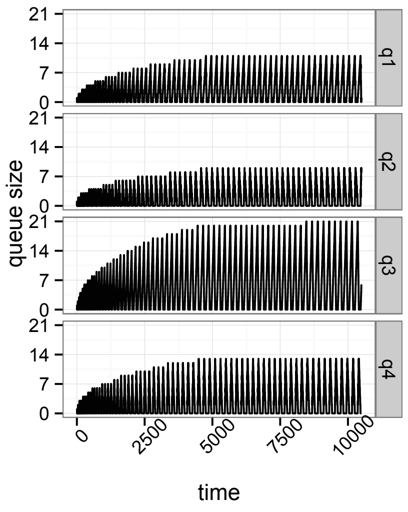
(a) *Example* 1.



(b) *Example* 2.



(c) *Example* 3.



(d) *Example* 4.

Figure 7.2: Evolution of queue sizes for the systems in Examples 1 to 4.

Algorithm 1 Utilization Proportional Resource Manager

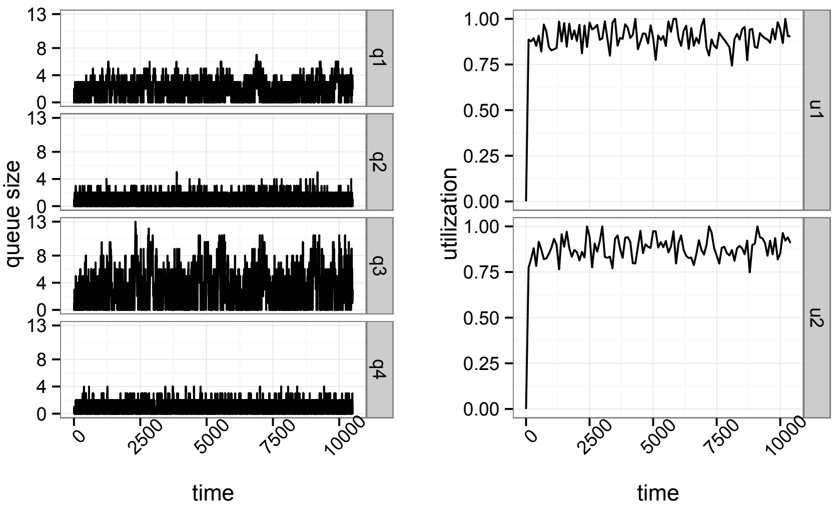
```
1: function P_CONTROLLER
2:     /* measure $U_{i[k]}, \forall i \in \mathcal{I}_{\mathfrak{R}}$ */
3:     $err_{[k]} \leftarrow U^{ref} - \max_i\{U_{i[k]}\}$
4:     $\vec{\rho}_{[k]} \leftarrow \vec{\rho}^{ref} + K_p err_{[k]} 1_n$
5:     return $\vec{\rho}_{[k]}$
6: end function
```

Algorithm 2 Utilization Proportional Resource Manager with Guarding

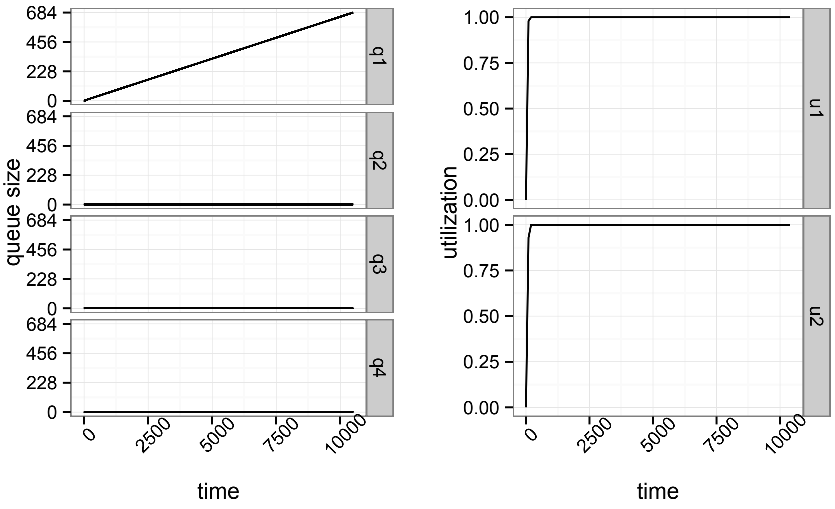
```
1: function GUARDED_P_CONTROLLER
2:     /* measure $U_{i[k]}, \forall i \in \mathcal{I}_{\mathfrak{R}}$ */
3:     /* measure $|\vec{x}_{[k]}|_p$ */
4:     if $|\vec{x}_{[k]}|_p < \Omega$ then
5:         return P_CONTROLLER() /* see Algorithm 1 */
6:     else
7:         return $\vec{\rho}^{\min}$
8:     end if
9: end function
```

portional controller that reacts to the maximum utilization on the resources. This resource manager chooses rates that deviate from ρ^{ref} depending on the error measured between the maximum utilization and U^{ref}. The resource manager presented in Algorithm 2 is a straight forward modification of Algorithm 1 by monitoring the norm of the state of the system and allowing the system to behave as in Algorithm 1 when the norm is below a bound, while selecting $\rho^{\min}$ otherwise.

Let us assume that the most desirable task chain rate in the system is $\rho^{ref} = 0.21$. Figure 7.3a presents the behavior of the system (queue sizes and processor utilization) when execution times of jobs vary randomly in their intervals and the rate of the task chain it fixed at ρ^{ref}. Since $\rho^{ref} \notin \Gamma_{\star}$, however, the system is unstable in the worst-case as we can see in Figure 7.3b. For the worst-case simulation we



(a) Average-case behavior.



(b) Worst-case behavior.

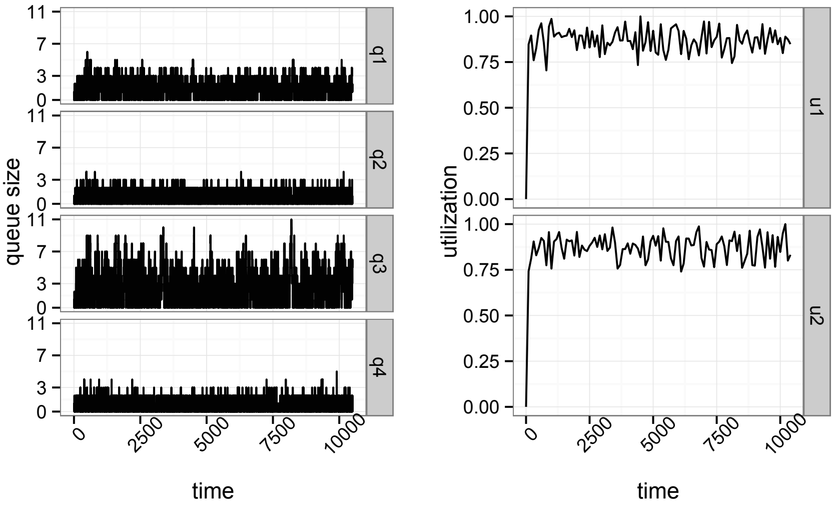
Figure 7.3: Behavior of the system when $\rho_{[k]} = \rho^{ref}$.

considered that execution times are constantly maximal.

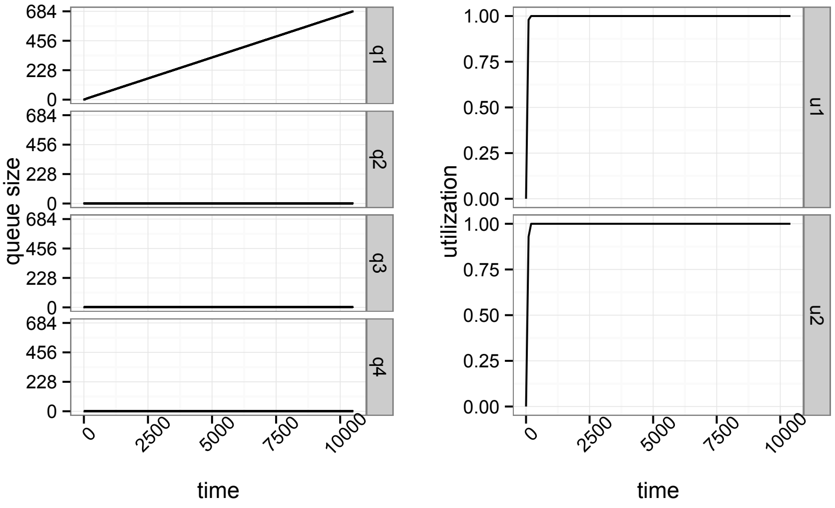
Example 5 considers a system which employs the resource manager presented in Algorithm 1 where the proportional coefficient is $K_p = 0.2$. Figure 7.4 presents the system's behavior. In the average-case, the behavior of the system is slightly better than when choosing a fixed rate at ρ^{ref} (queue sizes are slightly smaller while keeping the same utilization in the system). However, this system surprises us by being unstable in the worst-case. This happens because of the limitations of the resource utilization as a good measure of the state of the system. Since utilization saturates at 1, the measured error saturates at $U^{ref} - 1 = -0.1$. The smallest rate this controller can choose is $\rho^{ref} - 0.2 * (-0.1) = 0.19 \notin \Gamma_\star$ thus it doesn't satisfy our stability conditions from Theorem 3.

Example 6 attempts to correct the above behavior by choosing a better K_p coefficient. Since we need that $\rho^{ref} + K_p * (-0.1) \leq 0.125$ we obtain that $K_p \geq 0.85$. For this example we choose $K_p = 0.9$ and we present its behavior in Figures 7.5. From Figure 7.5b we can observe that the system remains stable in the worst-case. However, from Figure 7.5a we observe that the average-case performance of the system drops (queue sizes reach larger values than before). This happens because a large K_p coefficient implies too strong reactions to measured error in the system.

Example 7 presents a way to meet both requirements for average-case performance and worst-case stability, by employing the resource manager from Algorithm 2. For this system we select $K_p = 0.2$ as in *Example* 5, and we design the manager to stabilize with $\Omega = 2\frac{|o^{\max}|_p}{1-\mu} = 1.35$ ($o^{\max} = (8, 8, 0)^T$ for this system). The average-case and worst-case behaviors of this system are presented in Figures 7.6a and 7.6b respectively, and we can observe the desired behavior that we were after.

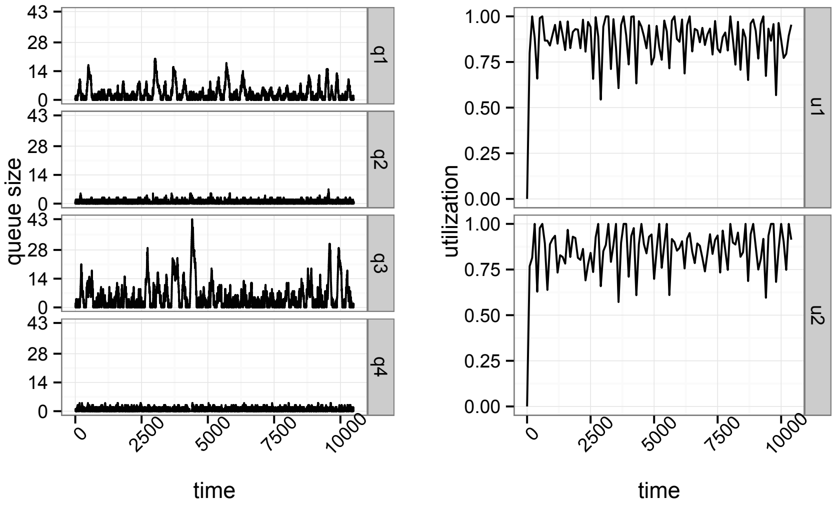


(a) Average-case behavior.

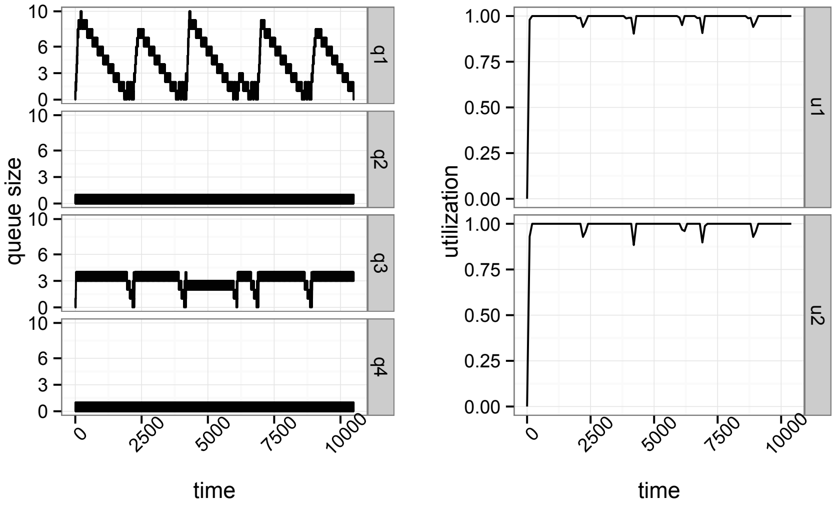


(b) Worst-case behavior.

Figure 7.4: Behavior of the system from *Example* 5.

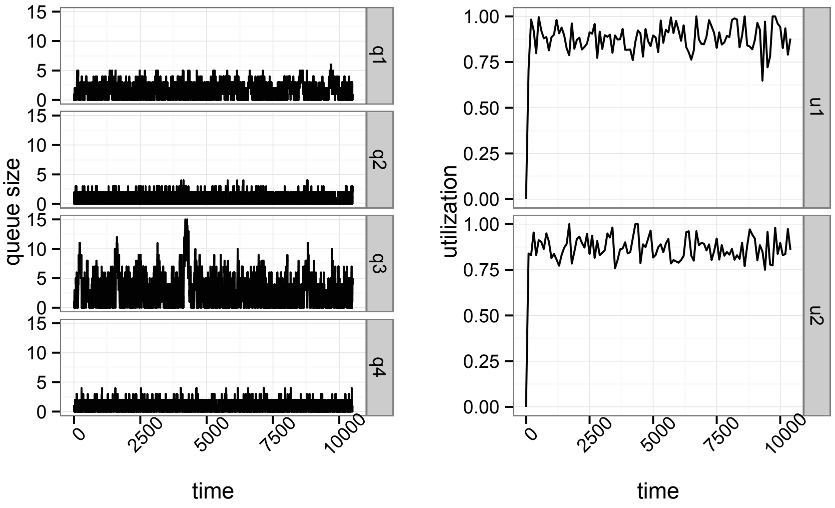


(a) Average-case behavior.

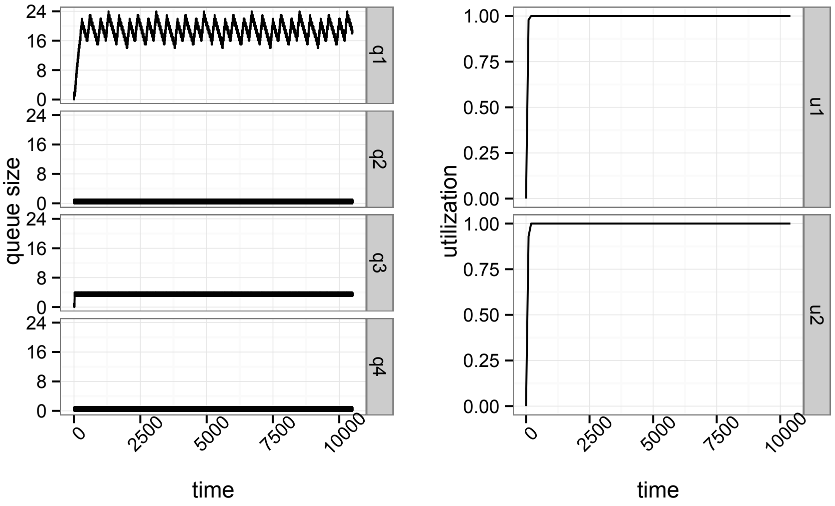


(b) Worst-case behavior.

Figure 7.5: Behavior of the system from *Example* 6.



(a) Average-case behavior.



(b) Worst-case behavior.

Figure 7.6: Behavior of the system from *Example* 7.

7.2 Stability of Uniprocessor Systems

In this section we will show how to apply our stability criterion for several resource managers. First we develop two simple ad-hoc resource managers and we show how they trivially satisfy our stability criterion and then we show how our stability criterion can be applied to three resource managers (*QRAM* [Raj97], *corner-case* [Raf10], and *QoS derivative* [Raf10]). All these resource managers were built for uniprocessor systems that run independent tasks (task chains formed of one single task). In this particular context the necessary conditions become sufficient as well, thus, we only need to satisfy Theorems 1 and 3.

7.2.1 Stability of Ad-Hoc Resource Managers

Let us assume that we have a system working as described in Section 3.2 for which the designer has knowledge about the expected execution times (noted c_j^e, $j \in \mathcal{I}_{\boldsymbol{\Theta}}$) of each task in the system. This system produces its desired runtime performance when tasks are running at certain known rates, noted $\vec{\rho}^e$, and these rates do not lead to increasing task queues, if execution times for jobs of tasks are the expected ones. Furthermore, worst-case execution times for all tasks are known, and there exists a set of rates $\vec{\rho}^{\min}$ which satisfies Theorem 1. For this system, one may imagine a resource manager which measures task queue sizes at certain moments in time and computes the norm. If the norm is larger than a predefined bound, the task chain rates are switched to $\vec{\rho}^{\min}$, otherwise, they are kept to $\vec{\rho}^e$. We can trivially show that this resource manager leads to a stable system according to Theorem 3. The algorithm for this resource manager is given in Algorithm 3. The qualitative transfer function for this resource manager is given in Figure 7.7a.

For the above described resource manager, it might happen, in practice, that its norm always remains around the bound, and there is a lot of switching between the two sets of task rates. To mitigate

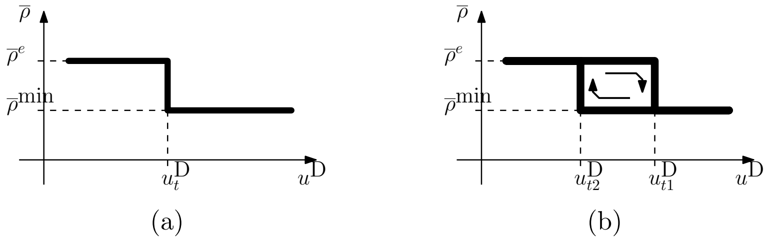


Figure 7.7: Qualitative transfer functions ($\vec{\rho}_{[k+1]} = f(\vec{x}_{[k]})$) of the resource managers presented in Algorithms 3 and 4.

Algorithm 3 Switching Resource Manager

```
1: function SWITCHING_CONTROLLER
2:     /* measure $q_{j[k]}, \forall j \in \mathcal{I}_\Theta$ */
3:     /* compute $|\vec{x}_{[k]}|_p$ */
4:     if $|\vec{x}_{[k]}|_p \geq U_t^D$ then
5:         return $\vec{\rho}^{\min}$
6:     else
7:         return $\vec{\rho}^e$
8:     end if
9: end function
```

Algorithm 4 Switching Resource Manager with Hysteresis

```
1: function HYSTERESIS_SWITCHING_CONTROLLER
2:     /* measure $q_{j[k]}, \forall j \in \mathcal{I}_\Theta$ */
3:     /* compute $|\vec{x}_{[k]}|_p$ */
4:     if $|\vec{x}_{[k]}|_p \geq U_{t1}^D$ and $\vec{\rho}_{[k-1]} = \vec{\rho}^e$ then
5:         $\vec{\rho}_{[k]} \leftarrow \vec{\rho}^{\min}$
6:     else
7:         if $|\vec{x}_{[k]}|_p \leq U_{t2}^D$ and $\vec{\rho}_{[k-1]} = \vec{\rho}^{\min}$ then
8:             $\vec{\rho}_{[k]} \leftarrow \vec{\rho}^e$
9:         else
10:            /* do nothing */
11:        end if
12:    end if
13:    return $\vec{\rho}_{[k]}$
14: end function
```

this problem the algorithm can be modified as in Algorithm 4 where $U^D_{t2} < U^D_{t1}$. The qualitative transfer function of this resource manager is given in Figure 7.7b and we can observe that it exhibits hysteresis. This is an even harder problem to analyze using established methods. However, it is still trivial to check, using our stability criterion, that this resource manager leads to a stable system according to Theorem 3.

7.2.2 Stability of the *QRAM*, *corner-case*, and *QoS derivative* Resource Managers

In this section, we take three resource management policies and determine if they lead to stable real-time systems. We describe the *QRAM*, the *corner-case*, and *QoS derivative* algorithms which have similarities in their functioning. They all:

- predict the expected future average execution times for jobs of tasks: $c^e_{j[k]}$, $j \in \mathcal{I}_{\mathbf{\Theta}}$,
- select task rates $\rho_{[k]}$ such that the expected resource demand:

$$U^e_{[k]} = \sum_{j \in \mathcal{I}_{\mathbf{\Theta}}} c^e_{j[k]} \frac{q_{j[k]} + \lceil \rho_{j[k]} \max\{0, h - \phi_{j[k]}\} \rceil}{h} \tag{7.1}$$

  equals the resource capacity: $U^e_{[k]} = 1$,
- use quality-of-service (QoS) functions of the form: $qual_j : \mathbf{P}_j \rightarrow \mathbb{R}_+$ which map the chosen rate to a quality value ($qual_j(\rho_j)$), and
- choose rates such that the total quality value $\sum\limits_{j \in \mathcal{I}_{\mathbf{\Theta}}} qual_j(\rho_j)$ is maximized.

The *QRAM* algorithm [Raj97] assumes concave, piecewise linear QoS curves and works by first selecting minimum rates for all tasks. As long as the expected resource demand falls below 1, it proceeds on choosing the task with the steepest quality to resource demand value and increases its rate until $U^e_{[k]} = 1$ or until it reaches a vertex where the slope of the curve changes. It then repeats the process with the next steepest slope.

The *QoS derivative* algorithm [Raf10] has the same assumptions and goal as the *QRAM* algorithm, however, it obtains lower complexity by assuming continuous concave QoS curves instead, and performs an optimization method similar to the steepest gradient method [Noc06].

The *corner-case* algorithms [Raf10] assumes continuous convex quality curves. It starts by computing the expected resource demand assuming minimal rates. It, then, determines what is the maximum rate from each task, that it can use in order to increase U^e to 1. It chooses to change the rate of the task that provides the biggest utility. If that is the maximum rate and the resource demand is still less than the capacity ($U^e < 1$) then the process is repeated for the remaining tasks in the system.

To prove that resource managers implementing the *QRAM* and *corner-case* algorithms satisfy the condition in Theorem 3 we simply observe that both algorithms start by choosing $\vec{\rho}^{\min}$ first, and than increase the rates while $U^e < 1$. Since $\vec{c}^e_{[k]} \geq \vec{c}^{\min}$, these algorithms will always choose $\vec{\rho}^{\min} \in \Gamma_\star$ if:

$$\frac{1}{h} \sum_{j \in \mathcal{I}_\Theta} c_j^{\min} \cdot q_{j[k]} \geq 1 \tag{7.2}$$

thus, we can determine that there exists a finite Ω:

$$\Omega \leq \max_{\sum_{j \in \mathcal{I}_\Theta} c_j^{\min} q_j = h} \left\{ \sum_{j \in \mathcal{I}_\Theta} c_j^{\max} q_j \right\}$$

and, thus, the proof is complete.

The *QoS derivative* algorithm works in a different way. At the beginning it determines the estimated resource demand in the system, assuming current rates ($\vec{\rho}_{[k]} = \vec{\rho}_{[k-1]}$ in (7.1)). Then, the manager solves a convex optimization problem, whose goal is to select new task rates such that some quality-of-service function is maximized, with the constraint that the expected resource demand with the new rates is equal with the amount of available resources equals 1. If a feasible solution (a solution that satisfies the constraint $U^e_{[k]} = 1$) exists, then we can prove stability, since it mimics the behavior of *QRAM*. However, if the constraint cannot be satisfied by any $\vec{\rho} \in \mathbf{P}$, it cannot be demonstrated that the selected rates will satisfy Theorem 3. A straight forward approach would be to test the solution of the optimization to determine if it is feasible, and if not, to select $\vec{\rho} \in \Gamma_\star$. However, for the convex optimization to find a feasible solution, it is required that the starting point ($\vec{\rho}_*$) satisfies the constraint[1], otherwise no feasible solution will be found [Boy08].

Algorithm 5 Modified *QoS derivative* Algorithm

```
 1: function MODIFIED_QOS_DERIVATIVE($\vec{c}_{[k]}$, $\vec{q}_{[k]}$, $\vec{\rho}_{[k]}$, $h$ )
 2:     /* compute $U^e_{[k]}$ assuming that $\vec{\rho}_{[k]} = \vec{\rho}_{[k-1]}$ */
 3:     while $j \leq n$ and $1 - U^e \neq 0$ do
 4:         $\rho_{j*} \leftarrow \max\left\{\rho_j^{\min}, \min\left\{\rho_j^{\max}, \frac{1-U^e}{c^e_{j[k]}} + \rho_{j[k]}\right\}\right\}$
 5:         $U^e \leftarrow U^e - c_{j[k]} \cdot \rho_{j[k]} + c_{j[k]} \cdot \rho_{j*}$
 6:         $j \leftarrow j + 1$
 7:     end while
 8:     $\vec{\rho}_{[k]} \leftarrow$ QoS derivative($\vec{\rho}_*$)
 9:     return $\vec{\rho}_{[k]}$
10: end function
```

Our solution is to modify the *QoS derivative* algorithm to start from a feasible point $\vec{\rho}_*$ as presented in Algorithm 5. The new initial rate vector $\vec{\rho}_*$ is computed line $3 - 7$. The original manager is then called (line 8 in the algorithm) with this rate vector, as a starting point. If there are $\vec{\rho} \in \mathbf{P}$ for which the constraint is satisfied, then

[1]In addition to this the Karush-Kuhn-Tucker matrix must be non-singular at the starting point, but it can be shown that this condition holds for any $\vec{\rho} \in \mathbf{P}$. See [Boy08] Sec. 10.2 for an in depth treatment of these conditions.

the algorithm has a feasible solution, otherwise, the algorithm will set $\vec{\rho}_{[k]} = \vec{\rho}^{\max}$ when the system is underloading and $\vec{\rho}_{[k]} = \vec{\rho}^{\min}$ when the system is overloading (condition (7.2)). From this behavior we can observe that the *QoS derivative* resource manager, with the above modification, is also stable.

7.2.3 Worst Case Response Time Bound

For any controller that satisfies the stability condition in Theorem 3 there exists a finite response time for each task. The actual value of the response time depends on the concrete scheduling policy and the controller ($f_c(\cdot)$) used in the system. In this section we will develop bounds on the worst case response time for tasks considering an EDF scheduler [Liu73] and two classes of controllers. The bounds developed here are different from the well known worst case response times derived in literature for EDF, since our system allows overload situations. The EDF scheduler considers as a working deadline for each job τ_{ij}, the sum of its release time and $1/\rho_{ij}$, where ρ_{ij} is the current job's rate. The bounds developed in this section are obtained assuming an uniprocessor system and are tighter than the simple bounds presented in Section 6.3.

For the following analysis, we will consider that the system always starts from a state where the queues are empty.

$$\mathcal{A} = \{\vec{x}_{[0]} \in \mathcal{X} | q = 0\} \tag{7.3}$$

In this case $|\vec{x}_{[0]}|_p = 0$, $\forall \vec{x} \in \mathcal{A}$ holds. According to equation (5.15), Ψ is the highest state norm ever achieved in the system. This result is important, since it allows us to bound the overload in the system. For the case of uniprocessors we have that $\zeta^{\max}_{[k]} = 0$, $\forall k$ and – since we only have one resource – $\vec{x}_{[k]} = |\vec{x}_{[k]}|_p = \sigma^{\max}_{[k]}$.

At a certain moment in time $t_{[k^\star]}$, a new job of a task $\tau_{j'}$ is released and we wish to compute its response time. We will denote this job with $\tau_{j'[k^\star]}$. In the system, at $t_{[k^\star]}$ there already exists a certain

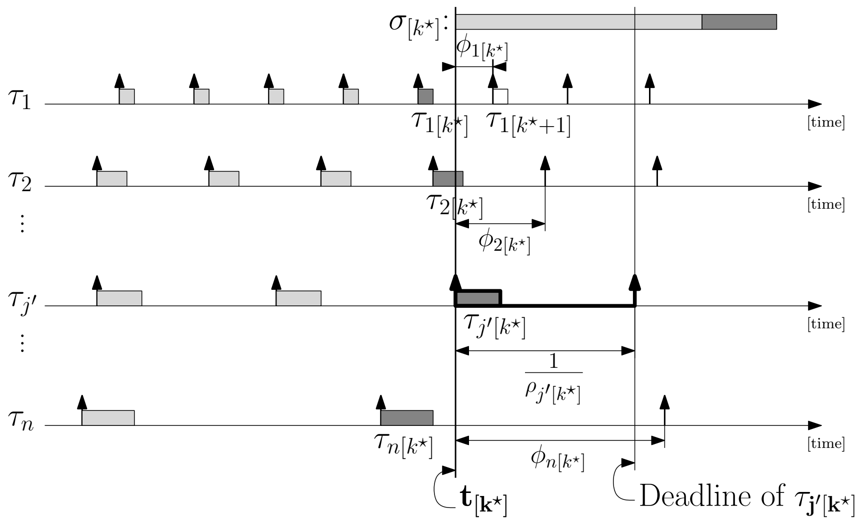


Figure 7.8: Accumulation of execution times

number of un-executed jobs and their total accumulation of execution times is:

$$\sigma_{[k^\star]} = \sum_{j \in \mathcal{I}_\Theta} c_{i[k^\star]} \cdot q_{i[k^\star]}$$

where $q_{i[k^\star]}$, $\forall i \in \mathcal{I}_\Theta$ are the queue sizes of each task and $c_{i[k^\star]}$ are the average execution time for the jobs in the queues (these averages are unknown, but $\vec{c}_{[k^\star]} \in \mathbf{C}$). Figure 7.8 illustrates this situation for a system of n tasks. All the jobs depicted in the figure are not yet executed at the moment $t_{[k^\star]}$, when $\tau_{j[k^\star]}$ is released. The dark shaded jobs represent the last released jobs of the tasks, just before $t_{[k^\star]}$. The light shaded jobs have been released before the dark shaded ones, and their deadlines are guaranteed to be prior to $t_{[k^\star]}$. Finally, all jobs with deadlines before the deadline of $\tau_{j'[k^\star]}$ (even the not yet released jobs such as $\tau_{1[k^\star+1]}$) must be executed before $\tau_{j'[k^\star]}$.

Since in overload situations EDF executes jobs non-preemptively, in the order of their working deadline, all light colored jobs in the figure will be executed before $\tau_{j[k^\star]}$, since their deadlines are before $t_{[k^\star]}$.

The execution times of these jobs represent $\sum_{j \in \mathcal{I}_\Theta} c_{i[k^\star]} \cdot (q_{i[k^\star]} - 1)$ out of $\sigma_{[k^\star]}$. From the rest of the jobs considered in $\sigma_{[k^\star]}$ (the dark colored ones), the ones with deadlines smaller than that of $\tau_{j[k^\star]}$ will be executed before it ($\tau_{1[k^\star]}$ and $\tau_{2[k^\star]}$ in the figure), and the rest will be executed after $\tau_{j[k^\star]}$ ($\tau_{n[k^\star]}$ in the figure). Also there may exist other, not yet released jobs, that will have their deadlines prior to the deadline of $\tau_{j[k^\star]}$ (e.g. $\tau_{1[k^\star+1]}$ in Figure 7.8) which also need to be considered. Taking all of this into account, and considering that $\rho_{i[k^\star]}$, $\forall i \in \mathcal{I}_\Theta$ are the release rates of all jobs, the response time of $\tau_{j[k^\star]}$ is:

$$r_{j'[k^\star]} = \sum_{j \in \mathcal{I}_\Theta} c_{j[k^\star]} \cdot (q_{j[k^\star]} - 1) + \sum_{j \in \mathcal{I}_\Theta} c_{j[k^\star]} \left\lfloor \frac{\frac{1}{\rho_{j'[k^\star]}} + \frac{1}{\rho_{j[k^\star]}} - \phi_{j[k^\star]}}{\frac{1}{\rho_{j[k^\star]}}} \right\rfloor \tag{7.4}$$

In the above formula, the quantities $(q_{j[k^\star]} - 1)$, $j \in \mathcal{I}_\Theta \setminus \{j'\}$ provide the number of non executed jobs of τ_j that have deadlines earlier than $t_{[k^\star]}$ while the quantities $\left\lfloor \frac{\frac{1}{\rho_{j'[k^\star]}} + \frac{1}{\rho_{j[k^\star]}} - \phi_{j[k^\star]}}{\frac{1}{\rho_{j[k^\star]}}} \right\rfloor$, $j \in \mathcal{I}_\Theta \setminus \{j'\}$ represent the number of released jobs of τ_j which have their deadlines between $t_{[k^\star]}$ and $t_{[k^\star]} + \frac{1}{\rho_{j'[k^\star]}}$ (the deadline of $\tau_{j'[k^\star]}$).

The following lemma gives an upper bound on the response time of the tasks in the system.

Lemma 1:
An upper bound on the worst-case response time of task τ_j in the system can be computed using the following equation:

$$r_j^{\max} = \frac{1}{\rho_{j'}^{\min}} + \Psi + \sum_{j \in \mathcal{I}_\Theta} c_j^{\max} \left\lfloor \frac{\rho_j^{\max}}{\rho_{j'}^{\min}} \right\rfloor \tag{7.5}$$

$\diamond$

PROOF The proof follows from equation (7.5) by observing that:

1. $\sigma_{[k^\star]} \leq \sigma_{[k^\star]} \leq \Psi$, and

2. the last released job of $\tau_{j'}$ was released with at most $1/\rho_j^{\min}$ before $t_{[k^\star]}$. ■

Up to this point, we have concerned ourselves with the scheduler used, and we have determined a formula for the worst-case response time for EDF assuming knowledge of the largest accumulation of execution times in the system (Ψ). One should note that Ψ typically depends on the scheduler and the controller (f_c) used in the system. The value computed in equation (5.15) is an upper bound on the largest state norm, since it is independent on these parameters. By adding extra constraints on the scheduler or the controller one may be able to tighten this bound. We will now consider two classes of controllers for which we will determine Ψ. The two classes of controllers are:

$$\mathbf{C1} = \left\{ f_c : \mathcal{X} \to \mathbf{P} \middle| \rho_{[k]} \in \begin{cases} \{\vec{\rho}^{\min}\}, & \text{if } \Gamma_{[k]}^{\alpha} = \emptyset \\ \mathbf{P}, & \text{otherwise} \end{cases} \right\} \tag{7.6}$$

$$\mathbf{C2} = \left\{ f_c : \mathcal{X} \to \mathbf{P} \middle| \rho_{[k]} \in \begin{cases} \{\vec{\rho}^{\min}\}, & \text{if } \Gamma_{[k]}^{\alpha} = \emptyset \\ \Gamma_{[k]}^{\alpha}, & \text{otherwise} \end{cases} \right\} \tag{7.7}$$

where $\Gamma_{[k]}^{\alpha}$ is

$$\Gamma_{[k]}^{\alpha} = \left\{ \vec{\rho} \in \mathbf{P} \middle| U_{[k]}^{e} = \sum_{j \in \mathcal{I}_{\Theta}} c_{j[k]}^{e} \frac{q_{j[k]} + \lceil \rho_j \max\{0,\ h - \phi_{j[k]}\} \rceil}{h} = \alpha \right\} \tag{7.8}$$

and $\alpha > 1$ is an arbitrary constant. $\Gamma_{[k]}^{\alpha}$ is the set of all rate vectors which will lead to $\sigma_{[k']} = h(\alpha - 1)$ (see equations (7.1)), where $t_{[k']} = t_{[k]} + h$.

The intuition behind the two classes of controllers is the following: **C2** always tries to take decisions such that σ is kept very aggressively around $h(\alpha - 1)$. When $\sigma_{[k]} \neq h(\alpha - 1)$, the accumulation will be brought back to $h(\alpha - 1)$ as soon as possible ($\sigma_{[k']} = h(\alpha - 1)$ if the prediction is correct). **C1** is a class of more general controllers, which includes **C2** as a particular case. We first show that both classes of controllers lead to stable systems.

Lemma 2:
Any system for which $f_c \in \mathbf{C1}$ is stable and $\mathbf{C2} \subset \mathbf{C1}$. $\diamond$

PROOF Since $\Gamma^{\alpha}_{[k]} \subset \mathbf{P}$ and $\{\vec{\rho}^{\min}\} \subset \Gamma_{\star}$, $\mathbf{C2} \subset \mathbf{C1}$ follows directly.

We will now show that for any system having $f_c \in \mathbf{C1}$, f_c also satisfies condition (5.14) for a certain Ω. We want to show that there exists a finite Ω such that, $\sigma^{\max}_{[k]} \geq \Omega$, implies that $\Gamma^{\alpha}_{[k]} = \emptyset$. We can then say that Ω is larger than the largest value of $\sigma^{\max}_{[k]}$ for which $\Gamma^{\alpha}_{[k]} \neq \emptyset$. We shall determine Ω by solving the following optimization problem:

$$\text{maximize } \sigma^{\max} = \sum_{j \in \mathcal{I}_{\Theta}} c_j^{\max} q_j \quad \text{subject to:}$$
$$U^e = \sum_{j \in \mathcal{I}_{\Theta}} c_j^e \frac{q_j + \lceil \rho_j \max\{0,\ h - \phi_j\} \rceil}{h} = \alpha$$

where $\vec{c}^e \in \mathbf{C}$, $\vec{\rho} \in \mathbf{P}$, and $[0]_n \preceq \vec{\phi} \preceq [1/\rho_j^{\min}]_n$. The solution is to maximize queue sizes, which happens when $\vec{c}^e = \vec{c}^{\min}$, $\vec{\rho} = \vec{\rho}^{\min}$, and $\vec{\phi} = [1/\rho_j^{\min}]_n$ and our optimization problem transforms itself into:

$$\text{maximize } \sigma^{\max} = \sum_{j \in \mathcal{I}_{\Theta}} c_j^{\max} q_j \quad \text{subject to:}$$
$$\sum_{j \in \mathcal{I}_{\Theta}} c_j^{\min} q_j = h(\alpha - \sum_{j \in \mathcal{I}_{\Theta}} c_j^{\min} \rho_j^{\min}) + \sum_{j \in \mathcal{I}_{\Theta}} c^{\min}$$

From the above we obtain that

$$\Omega = \max_{j \in \mathcal{I}_{\Theta}} \left\{ \frac{c_j^{\max}}{c_j^{\min}} \right\} \Big(h(\alpha - \sum_{j \in \mathcal{I}_{\Theta}} c_j^{\min} \rho_j^{\min}) + \sum_{j \in \mathcal{I}_{\Theta}} c_j^{\min} \Big) \tag{7.9}$$

and the proof follows. ■

Lemma 3:
For any system $\{\mathcal{T},\mathcal{X},\mathcal{A},\mathcal{S},\mathcal{U}\}$ with $f_c \in$ **C1** and $\mathcal{A}$ defined as in equation (7.3), the largest possible value of Ψ is given by

$$\Psi = \Omega + \sum_{j\in\mathcal{I}_\Theta} c_j^{\max} + h \sum_{j\in\mathcal{I}_\Theta} c_j^{\max}\rho_j^{\max} \tag{7.10}$$

where Ω is given by equation (7.9). $\diamond$

Proof Proof follows directly from equation (5.15). $\blacksquare$

Lemma 4:
For any $\{\mathcal{T},\mathcal{X},\mathcal{A},\mathcal{S},\mathcal{U}\}$ with $f_c \in$ **C2** and $\mathcal{A}$ defined as in equation (7.3), the largest possible value of Ψ is given by

$$\Psi = \max\left\{ \max_{j\in\mathcal{I}_\Theta}\left\{\frac{c_j^{\max}}{c_j^{\min}}\right\}h(\alpha-1),\ h\sum_{j\in\mathcal{I}_\Theta} c_j^{\max}\cdot\rho_j^{\max} + \sum_{j\in\mathcal{I}_\Theta} c_j^{\max} - h\right\} \tag{7.11}$$

$\diamond$

Proof We have two cases to analyze. When $\Gamma^\alpha_{[k-1]} \neq \emptyset$ then we have from equations (7.8), (7.7), and (4.5):

$$\sum_{j\in\mathcal{I}_\Theta} c^e_{i[k-1]} q_{j[k]} = h(\alpha-1)$$

The maximum accumulation in the system then becomes:

$$\begin{aligned}\sigma^{\max}_{[k]} = \sum_{j\in\mathcal{I}_\Theta} c_j^{\max} q_{j[k]} &= \sum_{j\in\mathcal{I}_\Theta} \frac{c_j^{\max}}{c^e_{j[k-1]}} c^e_{j[k-1]} q_{j[k]} \\ &\leq \max_{j\in\mathcal{I}_\Theta}\left\{\frac{c_j^{\max}}{c_j^{\min}}\right\} h(\alpha-1) \end{aligned} \tag{7.12}$$

On the other hand, when $\Gamma^\alpha_{[k-1]} = \emptyset$, there must exist a previous time instance $t_{[k-p]} \leq t_{[k-1]}$ with $\Gamma^\alpha_{[k-p+r]} = \emptyset$, $\forall r = \overline{0, p-1}$. In this case there are two possibilities: either $t_{[k-p]} = 1$ when

$$\sigma^{\max}_{[1]} = h\sum_{j\in\mathcal{I}_\Theta} c_j^{\max}\rho_j^{\max} + \sum_{j\in\mathcal{I}_\Theta} c_j^{\max} - h$$

or there exists $\Gamma^{\alpha}_{[k-p-1]} \neq \emptyset$ when inequality (7.12) applies and this concludes the proof. ■

The bound on the worst case response time (for an EDF scheduler) can be calculated using the equation (7.5) where Ψ is computed according with equation (7.10) for controllers in **C1**, and equation (7.11) for controllers in **C2**. Similar computations can be conducted for other classes of controllers as well.

The three resource managers described in the previous section (*QRAM*, *corner-case*, and *QoS derivative*) all belong to the **C2** class of controllers since they all predict future execution times and set rates such that the resource capacity is fully used under those conditions.

8 Conclusions and Future Work

In this chapter we conclude the thesis and discuss possible directions of future work.

8.1 Conclusions

In many real-time systems where variations in execution times are present and adaptation to these variations, in order to improve performance, is required, we face the issue of determining whether the adaptation mechanism is stable with respect to the timing properties of the system. Stability means that the state of the system remains bounded, implying bounded real-time properties such as worst-case response times and end-to-end delays.

In Chapter 4 of this thesis we have developed a detailed model describing the evolution in time of adaptive distributed real-time systems. We have then used this model to derive the worst-case behavior model of such systems. We ended the chapter with a discussion regarding the meaning and intuition of the parameters in the models.

In Chapter 5 we have used the worst-case behavior model of adaptive distributed real-time systems to derive stability conditions. We have developed necessary, and sufficient conditions (Theorems 1 and 2 respectively) for a system to have the possibility of being stable in the worst-case, and we have developed conditions that its resource manager must satisfy in order to stabilize the system (Theorem 3). Finally, we ended the chapter with an in-depth discussion about the geometric interpretation of these results.

In Chapter 6 we have addressed the limitations of our model and theory, and we have shown how our results can be extended to more general systems and actuation mechanisms. We have ended the chapter with a discussion linking our stability results with timing analysis.

Finally, in Chapter 7 we have shown, on several examples, how to apply the results developed in this thesis. We have shown how to prove stability for several resource managers. We have also shown how to modify a resource manager in order to achieve stability, and how to build stable resource managers. For the case of uniprocessor systems, we have also developed several results regarding their timing properties.

Although beyond the scope of this thesis, the framework presented here can also be used in an exploratory fashion in the design of distributed real-time systems. For example, we can:

- determine the necessary amount of adaptation (for example the needed minimum task graph rates) for stability, by using Theorem 1 to guide us,

- optimize the mapping of tasks to resources (where we could change the α coefficients, guided by Theorem 2), in such a way as to avoid unstable behaviors, and

- determine system requirements for stability (e.g. queue size), by determining the largest state of the system, with the help of

Theorem 3).

8.2 Future Work

The theory developed in this thesis forms a basis that has a large potential for improvement and extension.

Some specific problems which can be addressed are:

- Improving the approximations done when building the worst-case behavior of the system (inequality (4.16)). This problem is important as it directly affects the tightness of the obtained bounds. We have explained this issue in Sections 4.3, 4.3.3, and 4.5.

- Improving the gap between the necessary and the sufficient conditions for existence of resource managers that render the system stable. This problem directly affects the number of systems that can be guaranteed to be stabilized and, thus, needs addressing. We have provided ideas regarding the origins of this gap and how to approach it in Section 6.2.4.

- Improving the bounds on the state of the system. The bound on the state of the system (Ψ) emerges from the worst-case evolution of the system model that we use. Since we describe the behavior of a very large class of schedulers in our model, the worst-case behavior that we determine is pessimistic compared to the real behavior of the given system. We can improve the obtained bounds by modeling smaller classes of scheduling policies (see e.g. [Br96a, Br96b, But97, Bra01]).

- Diversifying the stability conditions (Theorem 3) with new results. The result presented in this thesis describes a switching controller. We have chosen this type of conditions as they are simple to test and because we believe that the effect of a piece of software code (the resource manager) can more easily be modeled as a switching mechanism than by other equations. Ideas about new kinds of stability conditions can be derived from the vast control theory literature currently existing (see e.g. [Ast97, Gla00, Oga70, Rug96, Sko96, Zho96]).

- Extensions to idling schedulers. This is a very important issue to address as, idling schedulers are extensively used in networks and buses, due to their deterministic nature and ease of implementation [Alm04]. Examples of very widely used idling bus protocols include TDMA [Bur10], FlexRay [Fle05], and FTT-ethernet [Ped02]. Ideas for these extensions can be obtain by looking at the timing analysis of the respective protocols.

- Integration of this theory with adaptive scheduling policies. Adaptive scheduling policies, such as resource reservation [Pa09b] techniques, are typically idling scheduling policies. However, these policies are based on mathematical modeling, thus, allowing for a tighter integration into the theoretical framework proposed here.

Bibliography

[Abd00] T. F. Abdelzaher. An Automated Profiling Subsystem for QoS-Aware Services. IEEE Real-Time Technology and Applications Symposium, 2000.

[Abd03] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance Control in Software Services – Using a Control-Theoretic Approach to Achieve Quality of Service Guarantees. IEEE Control Systems Magazine, vol. 23, pp. 74-90, 2003.

[Abe98] L. Abeni, G. C. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 4-13, 1998.

[Abe04] L. Abeni, G. C. Buttazzo. Resource Reservation in Dynamic Real-Time Systems. Real-Time Systems Journal, vol. 27, issue 2, pp. 123-167, 2004.

[Alm04] L Almeida, P Pedreiras. Scheduling within Temporal Partitions: Response-Time Analysis and Server Design. Proceedings of the 4th ACM International Conference on Embedded software, pp. 95-103, 2004.

[Ami06] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Real-Time Databases Supporting

Imprecise Computations. IEEE Transactions on Computers vol. 55, issue 3, pp. 304-319, 2006.

[And01] B. Andersson, S. Baruah, J. Jonsson. Static-Priority Scheduling on Multiprocessors. In Proceedings of the IEEE Real-Time Systems Symposium, pp. 193–202, 2001.

[Ast97] K. J. Åström and B. Wittenmark. Computer-Controlled Systems. Prentice Hall, 1997.

[Bak03] T. P. Baker. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In Proceedings of IEEE Real-Time Systems Symposium, pp. 120–129, 2003.

[Bao09] M. Bao, A. Andrei, P. Eles, Z. Peng. On-line Thermal Aware Dynamic Voltage Scaling for Energy Optimization with Frequency/Temperature Dependency Consideration. Design, Automation Conference (DAC), 2009.

[Bha10] K. Bhatti, C. Belleudy, M. Auguin. Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC), pp.184,191, 2010.

[Boy08] S. Boyd, L. Vandenberghe. Convex Optimization. Cambridge University Press, 2008.

[Br96a] M. Bramson. Convergence to Equilibria for Fluid Models of FIFO Queueing Networks. Queueing Systems vol. 22, pp. 5–45, 1996.

[Br96b] M. Bramson. Convergence to Equilibria for Fluid Models of Head-of-the-Line Proportional Processor Sharing Queueing Networks. Queueing Systems vol. 23, pp. 1–26, 1996.

[Bra01] M. Bramson. Earliest-due-date, First-Served Queueing Networks. Queueing Systems vol. 39, pp. 79–102, 2001.

[Bra08] M. Bramson. Stability of Queueing Networks. Springer, 2008.

[Bur98] A. Burns and D. Prasad, Value Based Scheduling of Flexible Real-Time Systems for Intelligent Autonomous Vehicle Control. Proc. Third IFAC Symp. Intelligent Vehicles, Elsevier Science, pp. 127-132, 1998.

[Bur10] P. Burgio, M. Ruggiero, F. Esposito, M. Marinoni, G. Buttazzo, L. Benini. Adaptive TDMA bus Allocation and Elastic Scheduling: a Unified Approach for Enhancing Robustness in Multi-Core RT Systems. IEEE International Conference on In Computer Design (ICCD), pp. 187-194, 2010.

[But97] G. C. Buttazo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Kluwer Academic, 1997.

[But98] G. C. Buttazo, G. Lipari, and L. Albeni. Elastic Task Model for Adaptive Rate Control. In Proceedings of the IEEE Real-Time Systems Symposium, pp. 286, 1998.

[But02] G. C. Buttazo and L. Albeni. Adaptive Workload Management through Elastic Scheduling. Journal of Real-Time Systems, vol. 23, pp. 7-24, 2002.

[But04] G. C. Buttazo, M. Velasco, P. Marti, G. Fohler. Managing Quality-of-Control Performance Under Overload Conditions. In Proceedings of the Euromicro Conference on Real-Time Systems, pp. 53-60, 2004.

[But07] G. C. Buttazzo, M. Velasco, P. Marti. Quality-of-Control Management in Overloaded Real-Time Systems. IEEE

Transactions on Computers, vol. 56, issue 2, pp. 253-266, 2007.

[Cas88] T. L. Casavant, J. G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. IEEE Transactions on Software Engineering, vol. 14, issue 2, pp. 141-154, 1988

[Cer02] A. Cervin, J. Eker, B. Bernhardsson, and K. E. Årzén. Feedback-Feedforward Scheduling of Control Tasks. Real-Time Systems, vol. 23, pp. 25-53, 2002.

[Ce03a] A. Cervin and J. Eker. The Control Server: A Computational Model for Real-Time Control Tasks. Proceedings of the 15th Euromicro Conference on Real-Time Systems, 2003.

[Ce03b] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K. E. Arzén. How Does Control Timing Affect Performance? Analysis and Simulation of Timing using Jitterbug and TrueTime. Control Systems, vol. 23, issue 3, pp. 16-30, 2003.

[Cer05] A. Cervin, J. Eker. Control-Scheduling Codesign of Real-Time Systems: The Control Server Approach. Journal of Embedded Computing, vol. 1, issue 2, pp. 209-224, 2005.

[Cha09] T. Chantem, X. S. Hu, and M.D. Lemmon. Generalized Elastic Scheduling for Real-Time Tasks. IEEE Trans. on Computers, vol. 58, no. 4, pp. 480–495, 2009.

[Chi06] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J.Stankovic, and T. Abdelzaher. Real-time power-aware routing in sensor networks. IEEE 14th International Workshop on Quality of Service, (IWQoS), pp. 83-92, 2006.

[Cof72] E. G. Coffman, R. L. Graham. Optimal Scheduling for Two-Processor Systems. Acta Informatica, vol. 1, pp. 200-213, 1972.

[Co05a] J. Combaz, J. C. Fernandez, T. Lepley, J. Sifakis. Fine Grade QoS Control for Multimedia Application Software. In Proceedings of the Design, Automation and Test in Europe, pp. 1038-1043, 2005.

[Co05b] J. Combaz, J. C. Fernandez, T. Lepley, J. Sifakis. Control for Optimality and Safety. In Proceedings of the 5th Conference on Embedded Software, 2005.

[Com08] J. Combaz, J. C. Fernandez, J. Sifakis, and L. Strus. Symbolic Quality Control for Multimedia Applications. Real-Time Systems, vol. 40, pp. 1-43, 2008.

[Cuc10] T. Cucinotta and L. Palopoli. QoS Control for Pipelines of Tasks Using Multiple Resources. IEEE Transactions on Computers, vol. 59, pp. 416-430, 2010.

[Dai96] J. G. Dai, G. Weiss. Stability and Instability of Fluid Models for Re-entrant Lines. Mathematics of Operational Research, vol. 21, pp. 115–134, 1996.

[Dai13] X. Dai. Some Criteria for Spectral Finiteness of a Finite Subset of the Real Matrix Space. Linear Algebra and its Applications, vol. 438, issue 6, pp. 2717-2727, 2013.

[Dau92] I. Daubechies, J. C. Lagarias. Sets of Matrices All Infinite Products of Which Converge. Linear Algebra and its Applications, vol. 161, pp. 227-26, 1992.

[Dau01] I. Daubechies, J. C. Lagarias. Corrigendum/addendum to: Sets of matrices all infinite products of which converge. Linear Algebra and its Applications, vol. 327, issues 1–3, pp. 69-83, 2001.

[Den97] Z. Deng, J. W. S. Liu, and J. Sun. A Scheme for Scheduling Hard Real-Time Applications in Open System Environment. In Euromicro Workshop on Real-Time Systems, 1997.

[Eke09] J. Eker. Multicore Scheduling Issues in Ericsson Mobile Platforms. International Conference on Parallel Processing Workshops, 2009.

[Fle05] FlexRay Consortium. FlexRay Communications System Protocol Specification Ver- sion 2.1. 2005.

[Fon11] D. Fontanelli, L. Palopoli, L. Greco. Deterministic and Stochastic QoS Provision for Real-Time Control Systems. Real-Time and Embedded Technology and Applications Symposium (RTAS), pp.103-112, 2011.

[Fon10] D. Fontanelli, L. Greco, L. Palopoli. Adaptive reservations for feedback control. IEEE Conference on Decision and Control (CDC), pp.4236-4243, 2010.

[Gho03] S. Ghosh, R. Rajkumar, J. Hansen, J. Lehoczky. Scalable Resource Allocation for Multi-Processor QoS Optimization. International Conference on Distributed Computing Systems, pp. 174, 2003.

[Gho04] S. Ghosh, J. Hansen, R. Rajkumar, J. Lehoczky. Integrated Resource Management and Scheduling with Multi-Resource Constraints. In Proceedings of the IEEE International Real-Time Systems Symposium, pp. 12-22, 2004.

[Gla00] T. Glad, L. Ljung. Control theory. CRC press, 2000.

[Gru67] B. Grünbaum. Convex Polytopes, Wiley, London, 1967.

[Gug05] N. Guglielmi, F. Wirth, and M. Zennaro. Complex Polytope Extremality Results for Families of Matrices. SIAM

Journal on Matrix Analysis and Applications, vol. 27, no. 3, pp. 721-743, 2005.

[Gug09] N. Guglielmi, M. Zennaro. Finding Extremal Complex Polytope Norms for Families of Real Matrices. SIAM Journal on Matrix Analysis and Applications, vol. 31, no. 2, pp. 602-620, 2009.

[Hal13] D. Hallmans, K. Sandstrom, M. Lindgren, T. Nolte. GPGPU for Industrial Control Systems. IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1-4, 2013.

[Har02] D. J. Hartfiel. Nonhomogeneous Matrix Products. World Scientific Publishing, 2002.

[Hen04] D. Henriksson, Y. Lu, T. Abdelzaher. Improved prediction for Web server delay control. In Proceedings of the Euromicro Conference on Real-Time Systems, pp. 61-68, 2004.

[Hu06] X. Hu, T. Chantem, M. Lemmon. Optimal Elastic Scheduling. In IEEE Real-Time and Embedded Technology and Applications Symposium, 2006.

[Izo08] V. Izosimov, P. Pop, P. Eleş, Z. Peng. Synthesis of Flexible Fault-Tolerant Schedules with Pre-emption for Mixed Soft and Hard Real-Time Systems. Proceendings of the Euromicro Conference on Digital system Design (DSD), pp. 71-80, 2008.

[Jia01] Z. P. Jiang, and Y. Wang. Input-to-State Stability for Discrete-Time Nonlinear Systems. Automatica, vol. 37, issue 6, pp. 857-869, 2001.

[Kan07] P.S. Kang, C. G. Lee. Search and Track Coordination in Multi-Ship Multi-Radar Systems using Schedulability Envelope. Real-Time Systems Journal, vol. 36, issue 3, pp. 227-262, 2007.

[Ke79] F. P. Kelly. Reversibility and Stochastic Networks. Wiley, New York, 1979.

[Kha11] N. M. Khalilzad, T. Nolte, M. Behnam, M. Asberg. Towards Adaptive Hierarchical Scheduling of Real-Time Systems. IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp.1-8, 2011.

[Kha13] N. M. Khalilzad, M. Behnam, T. Nolte. Adaptive Hierarchical Scheduling Framework: Configuration and Evaluation. IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp.1-10, 2013.

[Kop11] H. Kopetz. Real-time Systems: Design Principles for Distributed Embedded Applications. Springer, 2011.

[Kre89] E. Kreyszing. Introduction to Functional Analysis with Applications. John Wiley & Sons., Inci, 1989.

[Kum90] P. R. Kumar, T. I. Seidman. Dynamic Instabilities and Stabilization Methods in Distributed Real-Time Scheduling of Manufacturing Systems. IEEE Transactions on Automatic Control, vol. 35, pp. 289–298, 1990.

[Kum95] P. R. Kumar, S. Meyn. Stability Of Queueing Networks and Scheduling Policies. IEEE Trans. Automatic Control, 1995.

[Lee98] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On Quality of Service Optimization with Discrete QoS Options. In proceedings of Real-Time Technology and Applications Symposium, pp.276, 1998.

[Lee99] C. Lee. On Quality of Service Management. PhD thesis, Carnegie Mellon University, 1999.

[Liu73] C. L. Liu, J. W. Layland. Scheduling algorithms for multiprogramming in hard-real-time environment. Journal of ACM, pp. 40-61, 1973.

[Liu07] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. In Proceedings of the Conference on Decision and Control, pp. 3792-3799, 2007.

[Liu13] J. Liu, M. Xiao. Rank-one Characterization of Joint Spectral Radius of Finite Matrix Family. Linear Algebra and its Applications, vol. 438, issue 8, pp 3258-3277, 2013.

[Lu91] S. H. Lu, P. R. Kumar. Distributed Scheduling Based on Due Dates and Buffer Priorities. IEEE Transactions on Automatic Control, vol. 36, pp. 1406–1416, 1991.

[Lu02] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. Real-Time Systems, vol. 23, pp. 85-126, 2002.

[Lu05] C. Lu, X. Wang and X. Koutsoukos. Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks. IEEE Transactions on Parallel and Distributed Systems, vol. 16, pp. 550-561, 2005.

[Mar07] M. Marioni and G. C. Buttazo. Elastic DVS Management in Processors With Discrete Voltage/Frequency Modes. IEEE Transactions on Industrial Informatics, vol. 3, pp. 51-62, 2007.

[Mic08] A. N. Michel, L. Hou, D. Liu. Stability of Dynamical Systems: Continuous, Discontinuous, and Discrete Systems. Birkhäuser Boston, 2008.

[Ng02] J. K. Ng, K. Leung, W. Wong, V. Lee, and C. Hui. A Scheme on Measuring MPEG Video QoS with Human Perspective. Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications, pp. 233-241, 2002.

[Noc06] J. Nocedal, S. J. Wright. Numerical Optimization. Springer, 2nd edition, 2006.

[Nol09] T. Nolte, M. Behnam, M. Asberg, R. J. Bril, I. Shin. Hierarchical Scheduling of Complex Embedded Real-Time Systems. In Ecole d'Ete Temps-Reel (ETR'09), pp. 129–142, 2009.

[Oga70] K. Ogata, Y. Yang. Modern Control Engineering. 1970.

[Pal99] J.C. Palencia, M. Gonzãles Harbour. Exploiting Precedence Relations in the Schedulabiilty Analysis of Distributed Real-Time Systems. Proceedings of the IEEE REal-Time Systems Symposium, 1999.

[Pa09a] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari. AQuoSA – adaptive quality of service architecture. Journal of Software–Practice and Experience, vol. 39, pp. 1-31, 2009.

[Pa09b] L. Palopoli, L. Abeni. Legacy Real-Time Applications in a Reservation-Based System. IEEE Transactions on Industrial Informatics, vol. 5, issue 3, pp. 220-228, 2009.

[Ped03] P. Pedreiras, L. Almeida. The Flexible Time-Triggered (FTT) Paradigm: An Approach to QoS Management in Distributed Real-Time Systems. In Proceedings of the International Parallel and Distributed Processing Symposium, 2003.

[Ped02] P, Pedreiras, L. Almeida, P. Gai. The FTT-Ethernet Protocol: Merging Flexibility,Timeliness and Efficiency. In Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS), 2002.

[Rac06] R. Racu, R. Ernst. Scheduling Anomaly Detection and Optimization for Distributed Systems with Preemptive Task-Sets. Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 325-334, 2006.

[Raf10] S. Rafiliu, P. Eles, Z. Peng. Low Overhead Dynamic QoS Optimization under Variable Execution Times. In Proceedings of 16th IEEE Embedded and Real-Time computing Systems and Applications (RTCSA), pp. 293-302, 2010.

[Raf11] S. Rafiliu, P. Eles, Z. Peng. Stability Conditions of On-Line Resource Managers for Systems with Execution Time Variations. In the 23rd Euromicro Conference on Real-Time Systems (ECRTS), pp. 151-161, 2011.

[Raf13] S. Rafiliu, P. Eles, Z. Peng. Stability of Adaptive Feedback-based Resource Managers for Systems with Execution Time Variations. Real-Time Systems Journal, pp. 1-34, 2013.

[Raf–] S. Rafiliu, P. Eles, Z. Peng, and M. Lemmon. Stability of On-line Resource Managers for Distributed Systems under Execution Time Variations. Under submission.

[Raj97] R. Rajkumar, C. Lee, J. Lehoczky, D. Siewiorek. A Resource Allocation Model for QoS Mangement. Proceedings of the IEEE Real-Time Systems Symposium, pp. 298-307, 1997.

[Raj98] R. Rajkumar, K. Juvva, A. Molano, S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems. In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, 1998.

[Ryb92] A. N. Rybko, A. L. Stolyar. Ergodity of Stochastic Processes that Describe Functioning of Open Queueing Networks. Problems in Information Transmission, vol. 28, pp. 3–26 (in Russian), 1992.

[Rug96] W. J. Rugh. Linear System Theory. Prentice Hall, 1996.

[Sam08] S. Samii, S. Rafiliu, P. Eles, Z. Peng. A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems. In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 556-561, 2008.

[Set96] D. Seto, J. P. Lehoczky, L. Sha, K. G. Shin. On Task Schedulability in Real-Time Control Systems. In Proceedings of the 17th Real-Time Systems Symposium, 1996.

[Shi10] J. Y. Shi, A.N. An, K. Li, Y. Hao, P. Y. Liu, Y. Kang. Optimization of Shared Memory Controller for Multi-Core System. IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp.287,289, 2010.

[Sko96] S. Skogestad, I. Postlethwaite. Multivariable Feedback Control: Analysis and Design. New York, 1996.

[Son98] E. D. Sontag. Mathematical Control Theory: Deterministic Finite Dimensional Systems. Second Edition, Springer, New York, 1998

[Son01] E. D. Sontag. The ISS Philosophy as a Unifying Framework for Stability-Like Behavior. Lecture Notes in Control and Information Sciences, vol. 256, pp. 443-467, 2001.

[Spu94] M. Spuri and G. C. Buttazzo. Efficient Aperiodic Service under the Earliest Deadline Scheduling. In IEEE Real-Time Systems Symposium, 1994.

[Spu96] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. Real-Time Systems Journal, vol. 10, issue 2, 1996.

[Sri02] A. Srinivasan, S. Baruah. Deadline-Based Scheduling of Periodic Task Systems on Multiprocessors. Information Processing Letters, vol. 84, pp. 93–98, 2002.

[Tei12] R. Teichner, M. Margaliot. Explicit Construction of a Barabanov Norm for a Class of Positive Planar Discrete-Time Linear Switched Systems. Automatica, vol. 48, issue 1, pp. 95-101, 2012.

[Yao08] J. Yao, X. Liu, M. Yuan, and Z. Gu. Online Adaptive Utilization Control for Real-Time Embedded Multiprocessor Systems. In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, pp. 85-90, 2008.

[Zha06] Y. Zhang, K. Chakrabarty. A Unified Approach for Fault Tolerance and Dynamic Power Management in Fixed-Priority Real-Time Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 1, pp. 111-125, 2006.

[Zho96] K. Zhou, J. C. Doyle, K. Glover. Robust and Optimal Control. Prentice Hall, vol. 40, 1996.

Notations

Mathematical notations

$\mathbb{Z}_+$	Set of natural numbers	25
$\mathbb{R}$	Set of real numbers	81
$\mathbb{R}^n$	n-dimensional euclidian space	81
$\mathfrak{S}$	Set	25
$\mathcal{I}_{\mathfrak{S}}$	Index set of $\mathfrak{S}$	25
$\mathcal{P}(\cdot)$	Power set	25
$[a_{i,j}]_{n\times m}$	Matrix definition	25
I_n	Identity matrix	25
$0_{n\times m}$	Matrix with all coefficients 0	25
$[v_i]_n$	Vector definition	25
0_n	Vector with all coefficients 0	25
1_n	Vector with all coefficients 1	25
$\lvert\cdot\rvert$	Norm definition	26
$\mathrm{Ker}(A)$	Kernel space of matrix A	26
$\mathrm{Im}(A)$	Image space of matrix A	26
$A^2 = A$	Property of idempotent matrices A	26
$\succ, \succeq, \prec, \preceq$	Vector comparison operators	26

Distributed real-time system's parameters

$\mathfrak{R}$	Set of resources	26

n	Number of resources	26
$\mathcal{I}_{\mathfrak{R}}$	Index set of resources	26
$\mathbf{N}_i$	Resource i	26
$\mathcal{I}_{\mathbf{N}_i}$	Index set of tasks competing for $\mathbf{N}_i$	26
τ_j	Task j	26
$\mathfrak{A}$	Set of task chains	26
$\mathcal{I}_{\mathfrak{A}}$	Index set of tasks chains	26
$\mathfrak{C}$	Task chain	26
$\mathcal{I}_{\mathfrak{C}_p}$	Index set of task that are part of task chain $\mathfrak{C}_p$	27
$\mathbf{P}_p$	Set of instance release rates associated with $\mathfrak{C}_p$	27
$\mathcal{I}_{\boldsymbol{\Theta}}$	Index set of all task in the system	28
$\gamma(\cdot)$	Mapping from tasks to task chains	28
$\nu(\cdot)$	Mapping from tasks to resources	28
$\pi(\cdot)$	Mapping from tasks to the set of predecessors in their task chain	28
$\varphi(\cdot)$	Mapping from tasks to nodes that have their predecessors	28
$\mathfrak{p}(\cdot)$	Mapping from a task to its direct predecessor	29
$\mathfrak{p}'(\cdot)$	Mapping from a task to the set of its direct predecessors	93
τ_{jk}	kth job of task τ_j	29
c_{jk}	Execution time of job τ_{jk}	29
r_{jk}	Response time of job τ_{jk}	29
q_j	Queue size of task τ_j	30
Δ	Worst-case response time of the resource manager	30
h	The resource manager's actuation period	31

Control theory

$t_{[k]}$	The kth time moment in the system's evolution	37
$\mathcal{T}$	The discrete set of times at which the system evolves	31
$\mathcal{X}$	The set of states of the system	31
$\mathcal{A}$	The set of initial states	31

$\mathcal{U}$	The set of all possible inputs	31
$\mathcal{S}$	The set of all possible trajectories of the system	31
$(\mathcal{X}, \lvert\cdot\rvert)$	Normed space	31

System model

$c_{j[k]}$	Average execution time for jobs of τ_j executed between $t_{[k]}$ and $t_{[k+1]}$	40
$q_{j[k]}$	Queue size of task τ_j at time $t_{[k]}$	38
$_lq^{j'}_{j[k]}$	Link queue size associated with the dependency link from $\tau_{j'}$ to τ_j at time $t_{[k]}$	93
$s_{j[k+1]}$	Number of jobs that become ready for execution between $t_{[k]}$ and $t_{[k+1]}$	39
$\phi_{p[k]}$	Offset of task τ_j at $t_{[k]}$	39
$f_c(\cdot)$	Control law, implemented by the resource manager	41
$\sigma_{i[k]}$	Accumulation of execution times on resource $\mathbf{N}_i$, at $t_{[k]}$	41
$\sigma^{\max}_{i[k]}$	Upper bound on $\sigma_{i[k]}$ at $t_{[k]}$	42
$\zeta_{i[k]}$	Accumulation of execution times of released but not yet ready for execution jobs for tasks on resource $\mathbf{N}_i$, at $t_{[k]}$	42
$\zeta^{\max}_{i[k]}$	Upper bound on $\zeta_{i[k]}$ at $t_{[k]}$	42
$o_{i[k]}$	Accumulation of execution times determined by the offsets of task on resource $\mathbf{N}_i$, at $t_{[k]}$	42
$o^{\max}_{i[k]}$	Upper bound on $o_{i[k]}$ at $t_{[k]}$	42
$U_{i[k]}$	Prescribed utilization on resource $\mathbf{N}_i$, at $t_{[k]}$	42
$U^{\max}_{i[k]}$	Upper bound on $U_{i[k]}$ at $t_{[k]}$	43

Worst-case behavior model

$\alpha_{ii'}$	Coefficient of accumulation transfer from $\mathbf{N}_{i'}$ to $\mathbf{N}_i$	44
β_i	Worst-case accumulation due to partially executed jobs before arriving at $\mathbf{N}_i$	44

$l_{[k]}$	Branch of evolution of the system during $[t_{[k]}, t_{[k+1]}]$	45
A_l	Matrix parameters of the worst-case behavior model	45
$\vec{b}_l$	Vector parameters of the worst-case behavior model	46
$\vec{x}_{[k]}$	State of the worst-case behavior model at $t_{[k]}$ . . .	46
$\vec{w}_{[k]}$	Disturbance in the worst-case behavior model at $t_{[k]}$	46
$\vec{u}_{[k]}$	Input in the worst-case behavior model at $t_{[k]}$ . . .	46

Stability analysis

$\Gamma_\star$	Set of task chain rate vectors that keep all resources underloaded in the worst-case	63
$\lvert \cdot \rvert_p$	Norm of the system	66
$[p_i]_{n+1}$	Point in the state space, helpful in computing the norm	66
μ	Parameter helpful in determining $[p_i]_{n+1}$	66
B_l	Matrices derived from A_l, helpful in determining stability	66
Ω	Bound on the norm of the state	72
Ψ	Ultimate bound on the norm of the state	73

Definitions

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing – An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 1156 **Artur Wilk:** Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.

No 1183 **Adrian Pop:** Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.

No 1185 **Jörgen Skågeby:** Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.

No 1187 **Imad-Eldin Ali Abugessaisa:** Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.

No 1204 **H. Joe Steinhauer:** A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.

No 1222 **Anders Larsson:** Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.

No 1238 **Andreas Borg:** Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.

No 1240 **Fredrik Heintz:** DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.

No 1241 **Birgitta Lindström:** Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.

No 1244 **Eva Blomqvist:** Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.

No 1249 **Rogier Woltjer:** Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.

No 1260 **Gianpaolo Conte:** Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.

No 1262 **AnnMarie Ericsson:** Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.

No 1266 **Jiri Trnka:** Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.

No 1268 **Bahlol Rahimi:** Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.

No 1274 **Fredrik Kuivinen:** Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.

No 1281 **Gunnar Mathiason:** Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.

No 1290 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.

No 1294 **Johan Thapper:** Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.

No 1306 **Susanna Nilsson:** Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.

No 1313 **Christer Thörn:** On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.

No 1321 **Zhiyuan He:** Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.

No 1333 **David Broman:** Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.

No 1337 **Alexander Siemers:** Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.

No 1354 **Mikael Asplund:** Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.

No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.

No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish,2011,ISBN 978-91-7393-154-0.

No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.

No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.

No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control – Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.

No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.

No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.

No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.

No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.

No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.

No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.

No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.

No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.

No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.

No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.

No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.

No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.

No 1559 **Sergiu Rafiliu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.

Linköping Studies in Arts and Science

No 504 **Ing-Marie Jonsson:** Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.