From Competitive to Cooperative Resource Management for Cyber-Physical Systems

Lindberg, Mikael

2014

Document Version:
Publisher's PDF, also known as Version of record

From Competitive to Cooperative Resource Management for Cyber-Physical Systems

Mikael Lindberg

LUND
UNIVERSITY

Department of Automatic Control

PhD Thesis
ISRN LUTFD2/TFRT--1102--SE
ISBN 978-91-7623-018-3 (print)
ISBN 978-91-7623-019-0 (web)
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by MediaTryck.
Lund 2014

To Mirjam, Mattis and Rebecka

Abstract

This thesis presents models and methods for feedback-based resource management for cyber-physical systems. Common for the scenarios considered are severe resource constraints, uncertain and time-varying conditions and the goal of enabling flexibility in systems design rather than restricting it.

A brief survey on reservation-based scheduling, an important enabling technology for this thesis, is provided and shows how modern day resource reservation techniques are derived from their real-time system and telecommunications theory roots.

Techniques for modeling components of cyber-physical systems, including both computational and physical resources, are presented. The cyclic component model, specifically designed to model common resource demanding components in smart phones, is introduced together with techniques for model parameter estimation.

The topic of competitive resource management, where the different parts of the system compete for resources, is discussed using a smart phone platform as motivating example. The cyclic component model is used to form a rate-based performance metric that results in a convex optimization problem. A specialized optimization algorithm for solving this problem efficiently online and with limited precision hardware is introduced and evaluated through simulations.

A feedback control scheme for distributing resources in cases where components collaborate, i.e., where the performance metric is dependent on more than one component, is detailed and examined in a scenario where the available resource is limited by the thermal dynamics of the CPU. The scheme is evaluated through simulation of a conversational video pipeline. The thermal model is validated on a mobile robot, where it is used as part of an adaptive resource manager.

The problem of energy conservative distribution of content to a population of co-located mobile clients is used to motivate the chapter on cooperative resource management, i.e., scenarios where the participants have individual but similar goals and can benefit from sharing their partial results so that all collaborators save cost. The model for content trading is presented in synchronous and asynchronous formulations and performance is evaluated through both simulations and experimental results using a prototype implementation in an emulated environment.

Acknowledgements

It is safe to say that this thesis would not have existed if not for some of the amazing people I am privileged to be surrounded by. Combining my PhD studies with the busy life of fatherhood, and with my penchant for maintaining far too many hobbies and interests, has been a challenging and instructive journey, the completion of which I feel I must share some credits for.

Breaking off my career and returning to school for a PhD was not a decision I made lightly, and I feel very fortunate for the opportunity to do so. My thanks to Anders Robertsson for tricking me into this. I vividly remember the Thursday evening you called me and suggested I apply for the position, and I can assure all readers that there will be repercussions somewhere down the line.

Throughout my time at the department I have been able to rely on Eva Westin for support, friendship, advice and candy. She is an amazing person whom I hope will get all the recognition she deserves for what she does, both professionally and socially. Eva, my door is always open for you.

I wish to express my gratitude to my supervisor, Professor Karl-Erik Årzén, who has listened to my sometimes wild ideas and who has been a pillar of support during stressful moments of my studies. You have always had time for me when I asked for it and provided good directions, for research as well as for pubs.

To my co-supervisor, Johan Eker, I wish to extend my thanks for the discussions we had and for the much needed coaching during periods of doubt and uncertainty. Your levelheaded yet diligent approach to research has taught me many things I hope to incorporate into my own methods as I continue this path. And yes, I will probably buy that piano.

While I have been one of the older PhD students at the department, it has been comforting to always be put in place by the research engineers. Anders Blomdell, Leif Andersson and Rolf Braun, you have my thanks for your help, your dedication, and for constantly reminding me just how little I still know.

During my PhD years I have had many colleagues, some of which has become good friends. I would specifically like to mention Toivo Henningsson, my mentor at the department, whose enthusiasm and insights have inspired my work. I would also like to thank Anders Widd and Martin Hast for the conversations we had and

Karl Berntorp for paving the way during the thesis writing period. Coffee is on me from now on.

Though perhaps not consciously aware of their contributions, I would like to mention my children, Mattis and Rebecka, as some of the more important influences for my work. There is no finer way to study dynamic constrained resource management than observing our hallway in the morning as we are trying to get everybody off to school and work.

Last but not least, I would like to thank my wife Mirjam for her unwavering support during these years. Despite everything, I have never heard you regret once that we took this decision, even though my own resolve has sometimes wavered. There is no person to whom I owe more, nor any person I would rather have by my side for the next challenge.

Mikael

This research has partially been funded by the VINNOVA/Ericsson project "Feedback Based Resource Management and Code Generation for Real-time System" , the EU ICT project CHAT (ICT-224428), the EU NoE ArtistDesign, the Linneaus Center LCCC, the ELLIIT strategic research center, and the Ack'a VR project 2011-3635 "Feedback-based resource management for embedded multicore platforms".

Contents

1

Introduction

1.1 Background and motivation

Resource management considerations are increasingly shaping the development of new technology, be it embedded computers or large scale infrastructure systems, and the limits of realizable functionality is often defined by factors such as energy or available network bandwidth. The current global focus on designing energy efficient systems and ecologically sustainable technological growth is likely to emphasize these issues even further in the foreseeable future.

With that in mind, the central theme of this thesis is chosen to be resource management for computer systems. While resource management for embedded systems has for a long time been modeled as real-time scheduling problems, such methods are primarily used in monolithic systems with well known structure. This thesis chooses instead to focus on the highly dynamic and uncertain cases introduced by the increasingly advanced types of embedded systems developed today.

Systems controlled by software are inherently flexible and reconfigurable. Enabling system designers to use this flexibility while retaining control over system performance is an important goal of this work. Feedback, estimation and online optimization take the place of current methods based on prior knowledge, and the models employed are tailored towards a holistic point of view, where the inclusion of both physical and computational resource dynamics is necessary to describe system performance.

It is the intention of this work to propose new ways to view resource management for computer- and cyber-physical systems, so that their potential can be further explored. The resource management techniques provided by scheduling theory therefore form a part of the schemes discussed rather than being the focal point of them.

The methods presented draw upon several disciplines to provide a framework for resource management, including

- control theory,
- system identification,

- convex optimization,
- reservation based scheduling, and
- peer-to-peer technology.

The target systems are embedded or cyber-physical systems that are such that resource constraints and uncertainty would make worst case methods infeasible.

The thesis is based on the following publications:

Lindberg, M. (2007). *A Survey of Reservation-Based Scheduling*. Technical Report ISRN LUTFD2/TFRT--7618--SE. Department of Automatic Control, Lund University, Sweden.

This technical report is a survey on the state of the art concerning reservation-based scheduling. The survey covers the origins of the field as it appears both in real-time computing and telecommunication and how the two fields have merged in present day.

M. Lindberg was the author of the report and collected the information it was based on.

Lindberg, M. (2009). "Constrained online resource control using convex programming based allocation". In: *Proceedings of the 4th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2009)*.

This paper presents a model for resource reservations for smart phones, based on convex optimization. Included is also an algorithm aimed at limited precision hardware to solve the optimization problem in an efficient manner.

M. Lindberg formulated the model, designed and implemented the optimization algorithm, and conducted experiments in order to examine the performance of the algorithm. M. Lindberg was also the author of the paper.

Lindberg, M. (2010). "Convex programming-based resource management for uncertain execution platforms". In: *Proceedings of First International Workshop on Adaptive Resource Management*. Stockholm, Sweden.

The contributions of this paper consists of the implementation of a resource manager based on the allocation algorithm presented in the previous papers and the results of experiments where the performance of the system is studied under time varying conditions. The paper also presents a model for cyclic software components that can be used to describe many types of resource demanding applications.

M. Lindberg developed the component model, implemented the resource manager and performed the experiments in the paper. He is also the author of the publication.

Lindberg, M. (2010). "A convex optimization-based approach to control of uncertain execution platforms". In: *Proceedings of the 49th IEEE Conference on Decision and Control*. Atlanta, Georgia, USA.

This paper is a rewritten version of the previous paper aimed at am automatic control audience. The models are reworked to better fit within the control domain.

M. Lindberg is the author of this paper and did the textual revision as well as the adaptation of the resource models.

Lindberg, M. and K.-E. Årzén (2010). "Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties". In: *Proc. 31st IEEE Real-Time Systems Symposium*. San Diego, CA.

This paper presents a model for collaborative resource management for interconnected components as well as a model for thermal control of an embedded CPU through constrained resource allocation. The paper presents a case study of a conversational video pipeline, with continuous thermal dynamics for the CPU. A scheme for feedback control-based optimization of performance metrics is discussed together with simulation results.

M. Lindberg is the originator of the approach as well as the designer of the feedback control mechanism. M. Lindberg also developed the software used for simulations and is the author of this paper. K.E. Årzén contributed the problem introduction and background.

Romero Segovia, V., M. Kralmark, M. Lindberg, and K.-E. Årzén (2011). "Processor thermal control using adaptive bandwidth resource management". In: *Proceedings of IFAC World Congress, Milan, Italy*. Milano, Italy.

In this paper, thermal CPU control based on dynamic resource allocation is demonstrated on a mobile robot. The contributions consists of experiments to validate the thermal model used in the previous paper on the robot as well as test cases with synthetic tasks running on the robot under time varying conditions.

M. Lindberg contributed the thermal control approach and the basic idea for the experiment and authored the part of the paper concerning the thermal control. V. Romero-Segovia designed the resource reservation part of the experimental set-up and wrote the sections in the paper that covers this. M. Kralmark performed the experiments on the robot. K.E. Årzén provided valuable input and reviewed the publication.

Lindberg, M. (2013). "Feedback-based cooperative content distribution for mobile networks". In: *The 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. Barcelona, Spain.

The paper introduces a scheme for energy conservation for content distribution in mobile networks. The contributions consists of a definition of the cooperative mechanism through which co-located mobile devices can exchange data so that all parties save energy and a feedback-based algorithm for optimizing the trades in real-time. The approach is evaluated through simulations.

M. Lindberg formulated the cooperative mechanism, designed the heuristic feedback mechanism and developed the simulation software used for the experiments. M. Lindberg is also the author of the paper.

Lindberg, M. (2014). "Analysis of a feedback-based energy conserving content distribution mechanism for mobile networks". In: *Proceedings of IFAC World Congress 2014*.

This paper expands on the cooperative scheme with time varying device populations and studies the throughput of the system under changing conditions. The resulting findings are discussed together with a design example.

M. Lindberg developed the software used to study the expanded cooperative distribution problem. performed the experiments and created the design example. M. Lindberg is also the author of the paper.

Lindberg, M. (2014). "A prototype implementation of an energy-conservative cooperative content distribution system". In: *The 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems.* In submission.

A prototype implementation of the cooperative distribution scheme is presented together with a modified asynchronous formulation of the underlying system model. The implementation is validated through experiments carried out in an emulated environment using laboratory PCs.

M. Lindberg contributed the asynchronous model and developed the software used for the experiments. M. Lindberg carried out the experiments and is the author of the paper.

1.2 Contributions

This thesis contains the following contributions

- A model for resource allocation for rate-based software components is proposed. The model is suitable for many types of applications that typically

require the most resources in cyber-physical systems, such as media players, controllers and games.

- An algorithm for solving convex allocation problems suitable for embedded platforms has been developed. The algorithm is designed to be easily implemented in fix-point arithmetics and with a varying problem structure.
- A control scheme for software components with multi resource dependencies is proposed. The scheme is intended to facilitate the integration of the physical and the computational aspects of a system.
- Experimental results from using CPU bandwidth reservation for thermal control are detailed. Experiments were conducted on a mobile robot using a Linux-based operating system and standard Intel-based hardware.
- An energy and spectrum conserving cooperative content distribution mechanism for wireless mobile devices using barter-like data trade is proposed. The mechanism uses a feedback-based arbitration algorithm in order to be viable in highly dynamic cases.
- A prototype implementation of the aforementioned mechanism is presented together with experimental results.

1.3 Outline of the thesis

This thesis is organized into chapters as follows. In Chapter 2, the formal definition of the problem is given. Chapter 3 presents relevant and related research, particularly covering the enabling technology of reservation based scheduling. Chapter 4 presents related results on the topic of power- and energy management. Models and estimation techniques suitable for cyber-physical systems with uncertain parameters are introduced in Chapter 5. Chapter 6 discusses resource management in systems where components compete for resources. The collaborative perspective, where components contribute to a common performance metric, is then presented in Chapter 7. Chapter 8 discusses techniques for cooperative resource management, where system components tries to maximize individual performance through cooperating. Chapter 9 then concludes with discussion and future research.

2

Problem formulation

This chapter introduces the problems treated in the thesis. The domain of smartphones serves as a basis for deriving the formal definition. An example from the control domain is added to provide an example of a resource management problem where components contribute to a common performance metric, and in order to show that the resource management problem is not unique to multimedia applications. Finally, the smartphone example is expanded into the mobile cloud formulation, with multiple independent units, to exemplify a case where units with individual performance metrics can all benefit from sharing resources.

In the case of smartphones, special attention is given to the point of view of platform providers, who would like to increase system robustness without posing overly prohibitive requirements on applications. In this context, a platform is a hardware and software design that supplies base functionality and resources. Complete products are then constructed by adding components built with a development kit which is distributed together with the platform. The Google Android platform is a recent example [Android, 2014].

For the mobile cloud scenario, the phone platform provider serves a more peripheral role as an enabler rather than the main stakeholder. Here it is rather assumed that the primary parties are phone users, who would like to minimize data traffic fees and maximize battery usage, and service providers that want to reduce network congestion.

2.1 Example 1 — Smartphones

The first motivating example comes from the domain of smartphones or tablet devices. These consumer products are fully customizable through downloadable applications - "apps" - and are often used for processing-intensive tasks, such as media playback, video recording or games. The exact resource needs for such applications is hard to predict as the software behavior in these domains is highly data and user command dependent. Since video playback is one of the more demanding tasks, this will be used as an illustrative example.

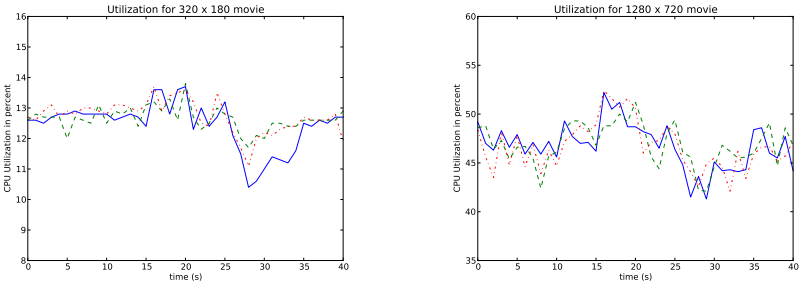


Figure 2.1 Resource requirements for decoding two versions of a H.264 movie on an Intel Core 2 Duo-based MacBook. The left plot represents a movie encoded in low resolution (320 by 180 pixels) while the right represents one encoded in HDTV-resolution (1280 by 720 pixels). The experiment was run three times with varying results, as illustrated by the the three curves in each plot. The utilization measure represents the percentage of time the decoding process had exclusive access to the CPU measured with a sliding 1 second time window.

Figure 2.1 exemplifies the CPU utilization for decoding two versions of the same video stream with perfect playback, i.e., no frame skipping or playback jitter. Note how the resource demands are significantly larger for the high resolution stream and how the levels change over time. There is also a visible trend, with a slight increase in resource demand around 15 seconds and a dip at around 28 seconds. This is evident for both streams and is caused by the encoding standard, which uses different levels of compression depending on the level of motion in the source video. The experiment was run three times for each stream with different results each time, this despite the fact that decoding a specific movie stream is a deterministic sequence of operations. One major reason for this is that modern hardware relies heavily on prediction and heuristics to minimize effects of memory latency and pipeline bubbles. Should these strategies fail, the system takes a performance hit. As the system doing the decoding in the example is executing a large number of background tasks in addition to the decoder, system state will vary from run to run.

The problem stated in traditional real-time terms would be to check the schedu-

lability of a set of periodic tasks $\tau_0, ..., \tau_N$ with the corresponding periods $T_0, ..., T_N$ and worst case execution times $C_0, ..., C_N$. Assume for simplicity that each task has a relative deadline equal to its period. If the scheduling policy used is Earliest Deadline First (EDF) and the system is a single core machine, the task set is schedulable if

$$\sum_{i=0}^{N} \frac{C_i}{T_i} \leq U_b \tag{2.1}$$

where U_b is the utilization bound, which depends on parameters such as the cost of context switches and is normally close to 1. For a media player task, the period would be equal to the frame rate at which the movie is encoded, which is easily accessible from the stream meta data.

If the test passes, deadlines will be met and the system performs as intended. If the test fails, the system is overloaded and tasks will miss their deadlines. Traditionally, a system should not admit tasks that will cause overload, but given the uncertainties mentioned above, it is not clear if enough information would be available to make such decisions. Specifically,

- the set of active tasks will change over time as the user enables different applications,
- the resource requirements of a task can vary greatly depending on input, and
- the properties of 3rd party software might not be available during system design.

A system designer could choose to restrict the use cases supported by the device in order to counteract some of these points, but this could render the product unattractive to consumers. It is also probable that a user would rather have access to a function running with degraded performance than being denied this functionality completely. Therefore, the all-or-nothing property of hard real-time formulations is not suitable for this problem domain.

This thesis chooses to focus on the following aspects of the problem:

- **Uncertainty in hardware and software.** The reliance on prior information in traditional real-time systems is increasingly a bottleneck in designing feature-rich embedded systems. Rather, the approach taken here will be to try to model the resource consumption of a system and estimate model parameters online.

  This has the effect of reducing the work needed by both hardware designers and software developers, thereby reducing time to market for both product and 3rd party add-ons.

- **Allocation under overload conditions.** For portable devices it is desirable to use low power components (CPUs, batteries, radio transmitters etc) and as a

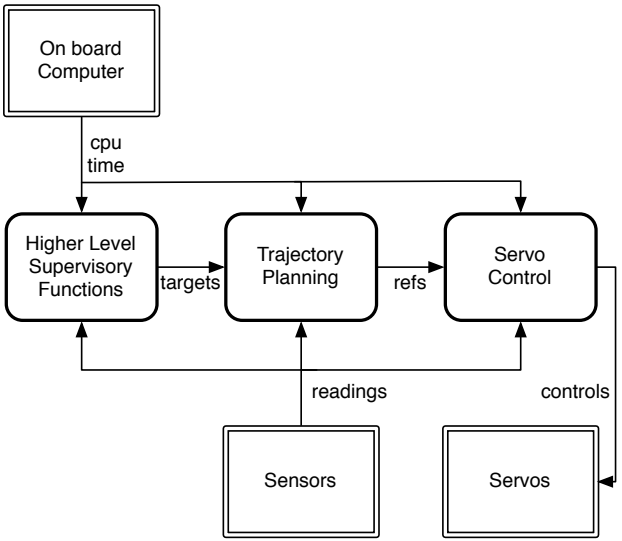


Figure 2.2 A schematic over a simple mobile robot system. The square blocks represent hardware functions that require power, the round-corner shapes are software functions that require cpu-time to run. The complex dependency situation makes it non-trivial to determine how to prioritize in a situation with insufficient resources to run the system at nominal performance.

result, efficient systems will often run near or in overload conditions in order to save power and unit cost.

The thesis will strive to provide resource management strategies that effectively manage systems under both nominal and overload conditions.

- **Non-restrictive assumptions on software components.** One way to simplify the work of the system designer would be to shift some of the burden to the 3rd party developers, e.g., requiring them to supply worst case execution time estimates and other detailed resource demand information. As such figures require technical expertise with the target platform, this could impede the supply of attractive 3rd party software.

  The application framework should put clear but lenient requirements on developers in order to make the platform simple and attractive to develop for.

2.2 Example 2 — Mobile robotics

Mobile robotics is another field where resource management is key to performance. Not only are computational resources scarce, there are usually many subsystems

besides the computer that compete for power. The drive to increase autonomy, and include more and more functions while simultaneously pricing products competitively, is making the resource constraints more and more pronounced.

As the constraints grow increasingly severe, it becomes interesting to see if a holistic view on resource management can improve operational range or enable units to be built with cheaper components. By combining hardware and software models, a hybrid system description that is popularly referred to as a cyber-physical system (CPS) emerges. The objective here is to study the interactions between hardware and software and through this learn how global system performance is affected by the dynamics of both.

Cyber-physical systems, particularly in the mobile robot case, share many properties with the smartphone device class discussed previously. In particular, this includes a tolerance for degraded performance in subsystems. This makes it an interesting domain for studying resource allocation trade-offs.

Consider the schematic presented in Figure 2.2. The functions in a robot can be realized in hardware or software, making the resulting dependency graph include connections both from hardware to software and vice versa. In order to handle a situation where some resource is scarce, a model that can express the total performance dynamics of the system would be useful. It is the aim of this thesis to present some initial thoughts on how this can be accomplished.

The thesis will focus on the following aspects of the problem:

- **Cyber-physical dynamics.** The coupling between computations and physics becomes apparent in mobile robotics, especially in constrained resource cases. The computational side will traditionally apply techniques similar to those discussed in Section 2.1 to decouple the dynamics by interfacing a perfectly timed real-time system with a sampled continuous time system through discrete time system descriptions with a predetermined sample period. Maintaining that sample period is resource expensive as it requires worst case design.

  As with most mobile devices, limited cooling capabilities can force CPUs to reduce their performance to prevent overheating, introducing a variable amount of computational capacity in the same way as for the smart phone example. Dealing with - or even controlling - the heat generated by the system and the effects on resource availability this entails is therefore imperative.

- **Distributed component systems.** Unlike the smartphone, the mobile robot is more modular. Sensors, actuators and computer components are spread out over a larger physical area, complicating things like synchronization and data flow management. Where in a monolithic computer, synchronization between processes is done through programmatic methods, in a mixed setting of physical and computational components such methods are not necessarily possible.

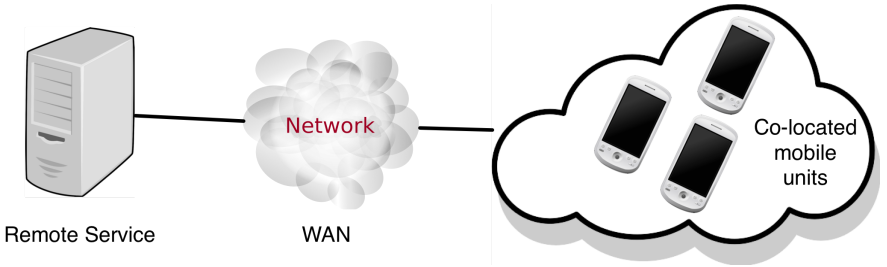


Figure 2.3 The use of cloud computing services, such as remote data storage or media streaming, by mobile devices is very natural and in some cases even necessary in order to deal with limited built in storage and sandboxed applications. What has yet to happen is the inclusion of the mobile devices as part of the cloud in the form of service providers.

2.3 Example 3 – Mobile cloud computing

The intrinsic limitations of mobile devices in terms of storage or computational capacity have made these devices heavily dependent on cloud computing services, such as remote storage or media streaming. With the rapid growth of both devices and services, it is perhaps not surprising that licensed spectrum is already scarce [FCC, 2010].

In order to reduce network load, it is tempting to implement peer-to-peer or peer-assisted mechanisms popular in fixed networks with stationary computers, such as BitTorrent, but the mobile case presents additional challenges before these technologies can be deployed. The limited energy available to a mobile device makes it inherently unattractive to help distribute data unless guarantees can be given that such investments will pay off.

By creating cooperative mechanisms where mobile devices can share the task of fetching data over expensive remote links, without risk of losing energy on the transaction, the mobile part of the network can be turned into willing service providers.

This thesis discusses how the coupling of resource constraints can lead to such schemes, where clients cooperate to solve problems and save resources while simultaneously reducing load on the Wide Area Network (WAN). The specific situation studied is cooperative file distribution to a population of co-located devices, under assumptions of high mobility and that nobody is willing to act altruistically.

Specifically, the thesis focuses on the following elements of the problem:

- **Cooperative sharing of resources** Finding ways to create win-win cooperative scenarios is important for systems of independent units, as altruistic behavior cannot be expected.

- **Dynamic populations** As in the case of the individual smart phones, where

the active software components will vary over time, the population of mobile devices interested in cooperating will vary over time and space. As such it is important that the mechanism uses feedback and estimation to make decisions and not rely on offline calculated solutions.

2.4 Problem features

A number of structural similarities present themselves in the above examples.

Limited resources

System performance is clearly limited by the availability of resources. Reducing size while adding functionality has been the trend in consumer electronics for a long time, so though components become smaller and more efficient, resource constraints continue to be an important factor in product design.

Uncertainty and estimation

Demand and supply of resources are expected to change over time, either spontaneously or as a function of how they are used. An example of the former is when components activate or switch modes, where the latter can be a result of the CPU heating up or a battery being drained. It is important to note that not only are system parameters expected to change over time, but the structure of the system (e.g., what components are active) is subject to continuous change as well.

Feedback and optimization

Rather than relying on pre-calculated solutions, the thesis uses optimization- and feedback-based techniques to continuously maximize the system utility, given the current system state and resource availability.

Taking decisions on a budget

In order to make continuous resource allocation decisions viable, the results need to be computed in a timely and resource-light fashion. Many powerful forms of optimization are therefore infeasible for this problem domain, due to heavy computational requirements and/or computational complexity that is difficult to predict.

2.5 Problem structures

In this thesis, the problems of resource management will be divided into three categories.

Competitive

In a competitive scenario, the components of the system have their own private performance metrics, which are functions of the component's resource allocation. In general, it is up to an outside party to determine the value of each component in relation to the others, something that is traditionally done in software systems through priority assignment by an outside party, often a system designer or user.

The thesis will generally consider under-provisioned cases, i.e., where there is insufficient resources to saturate the needs of all components. As the competitive case concerns systems with unrelated components, one problem is how a system designer can describe the desired compromise and how component designers can express the resource needs of their components in a way that is simple and yet robust to platform changes.

Collaborative

In collaborative systems, components have no individual performance metrics, but contribute to one or more system wide metrics. These metrics typically concern things like synchronized operation, end-to-end latency over a number of components, and buffer length control.

In over-provisioned cases, these types of problems are often formulated as synchronization and mutual exclusion problems, but as locking is problematic in resource management situations, the thesis offers a different view that can be applied to general CPS systems where functionality is implemented across both hardware and software.

Cooperative

Cooperative systems consist of components with individual performance metrics, but where they can benefit from exchanging data and services. The central theme here is not to allocate resources, but rather to allocate the spending of it so that the return on investment is as high as possible. This thesis will analyze an example of such a scheme where mobile clients cooperate around fetching data from a remote service and share it locally, thereby reducing both licensed spectrum and energy usage.

2.6 Overall goals

The objective of this work is to investigate how resource management can be done in situations where uncertainty in both demand and supply makes static methods infeasible. An effort is made to consider systems where components have dependencies, as this topic is less studied. While initially the work was focused on CPU resources inside a computer, the robot example makes it evident that if the availability of the CPU resource depends on the dynamics of other resources, a model encompassing both domains is desired.

The viewpoint is then shifted to encompass a population of devices that, while having individual goals, can cooperate in such a way that all participants benefit. The intent is to design methods for resource conservation where there is a tangible incentive to cooperate for everybody and not rely on assumptions of altruistic behavior.

It is an explicit goal that the methods presented are realistically implementable on power constrained systems and as such cannot be too resource consuming in themselves.

3

Reservation based scheduling

In order to state the design problems associated with embedded systems in resource management terms, the access to hardware and software resources must be exposed to the system designers as divisible resources. Resource access in computer systems has traditionally been posed as scheduling problems, but through virtualization and Reservation-Based Scheduling (RBS) techniques the access problem can be restated in resource allocation terms.

This chapter introduces the prior research related to RBS that is necessary to understand the structures for the competitive, collaborative and cooperative resource management strategies discussed in this thesis. Central concepts are introduced along with a brief a historical survey of RBS. The theory is derived from both its real-time scheduling roots and its queueing theory counterpart.

Using the RBS formulation, a resource management policy can be posed as the solution to an optimization problem. Such methods are introduced in the section on optimal resource management, which has its roots in operations management. It is worth noting that these techniques are mostly feed-forward in style, i.e., the allocations are calculated based on models rather than on-line measurements. Some important works in the alternative branch, feedback and adaptation, are then discussed.

As this thesis will largely be about on-line strategies, finding ways to solve optimization problems reliably and efficiently is a central part. The domain of convex optimization lends itself to this type of formulations and some examples of this are introduced in Section 3.4.

The chapter is based on the technical report [Lindberg, 2007].

3.1 Important concepts

This section introduces important concepts that will be used in the presentation of relevant research.

Temporal Isolation

A highly desirable property of the RBS approach is that a task that has reserved a specific amount of a resource should have access to this regardless of what other tasks are running on the system. This is called temporal isolation and makes very good sense for both continuous media and control type applications used as examples so far. Memory protection ensures that applications will behave functionally correct even in the presence of other, potentially malfunctioning, software. Temporal isolation provides a similar guarantee for temporal correctness.

Components and composition

In order to handle the complexity of large systems, the ability to gather parts into component structures that are closed under composition is vital. Threads with priorities, the building blocks of traditional operating systems, do not compose [Lee, 2006]. By using hierarchical RBS techniques, it is possible to enforce temporal isolation and thereby create groups of threads with essentially the same outside properties as the atomic thread. This enables component-wise testing and verification, and also removes the need to explicitly know the structure of 3rd party software.

Timing sensitive applications

In real-time situations, the timely completion of tasks is important. Normally, if a task has a real-time deadline, it is assumed to function nominally if the deadline is met and fail if the deadline is missed.

In soft real-time problems, deadlines are allowed to be missed occasionally and for applications in this domain it is interesting to discuss how the performance is affected by this. Applications where the performance depends on how well the deadlines are met are called *timing sensitive applications*.

Graceful QoS degradation

While it is possible to create an admission policy that denies tasks that would make the scheduler unable to sustain reservations, this might not be desirable from a user perspective. For consumer applications, it can be preferable to have a slight (and predictable) degradation in QoS as opposed to being denied starting applications altogether. This becomes even more evident in embedded systems where resources are scarce. Consider a mobile phone user engaged in a video conference call when a SMS message comes in. Most would be content to have some slight degradation in video quality while still being able to accept the SMS message. If the playback application in question is designed to be aware of its resource allocation, it can be assigned lower QoS in an as graceful way as possible.

3.2 Reservation Based Scheduling

This form of scheduling is used together with a class of real-time applications whose quality of output depends on sufficient access to a resource over time. Such applications are difficult to handle in terms of traditional hard real-time theory. The typical situation involves some type of continuous media task (playback or encoding), and it was in fact the need for support for media software that ignited interest in the field. This was in the early '90s when computers started to make their way into mainstream media production and consumption. While this remains the favored use case also today, other forms of computing can also benefit from RBS. This includes classes of systems that have traditionally been considered hard real-time. Before discussing the different algorithms for RBS, the problem background will be presented in more detail.

Origins

The case for Reservation Based Scheduling (RBS) was perhaps most famously made by Mercer et al in [Mercer et al., 1994]. The paper discusses processor reserves as a way to describe computational requirements for continuous media type applications and some challenges when this is implemented on a microkernel architecture.

The basis of the analysis is periodic tasks, characterized by execution time C and period T. Mercer observes that C is likely difficult to compute and suggests that the programmer supplies an initial estimate and that the scheduler then measures and adjusts the estimate (a feedback scheduling technique). The paper also introduces the concept of task CPU percentage requirement $\rho = C/T$ and the expected execution time of a task running at rate ρ as $D = C/\rho$. It is worth noting that these definitions are very close to what present day theory refer to as *bandwidth* and *virtual finish time* respectively. Although [Mercer et al., 1994] is frequently cited, many of the aspects of resource reservations and continuous media had already been discussed in earlier works.

Herrtwich presents a number of insights around the problem in [Herrtwich, 1991]. Like in [Mercer et al., 1994], the use of conventional scheduling schemes is deemed as inefficient and perhaps not serving the user needs. Herrtwich also brings up the importance of preventing ill-behaved applications from disturbing others (temporal isolation) and that the user might prefer graceful degradation of QoS to being prevented from starting new applications when the system is overloaded.

Herrtwich paper quotes heavily from the even earlier work [Anderson et al., 1990], which details how media type applications can be served by a resource reservation scheme based on preemptive deadline scheduling. The concept of resources is here extended to include not only CPU but also disk, networking and more. [Anderson et al., 1990] presents more theory, but lacks some of the more general insights in Herrtwich's work.

Taking a queue from telecommunication

In what seems like unrelated work, the telecommunications society was around this time investigating queuing algorithms which, it would turn out, share properties with process scheduling problems. [Demers et al., 1989] discusses the sharing of a link gateway between clients using the Weighted Fair Queueing algorithm (WFQ) citing "protection from ill-behaved sources", essentially temporal isolation, as one of its main advantages.

The central idea of the algorithm is to schedule the jobs in the order they would have been completed by a Weighted Round-Robin (WRR) scheduler, perhaps the most well known form of Time Division Multiple Access (TDMA) used in channel sharing. The job finishing times, though not named as such in this paper, are in subsequent works called *virtual finish times*. In other words, the scheduler decisions are based on how the task set would behave if each task was running on a private platform with a fraction of the actual system speed.

In this manner, the fairness property of the WRR scheduler and the finishing order of the jobs are preserved while the context switching overhead is reduced. Apart from being one of the earliest examples of temporal isolation, it introduces the notion of basing the scheduling decisions on virtual time metrics.

Virtual time

The WFQ scheme and how virtual time can be used is discussed in papers in the decade following [Demers et al., 1989]. One of the more comprehensive is [Parekh and Gallager, 2007], which further investigates how a Generalized Processor Sharing scheme (GPS) can be approximated using virtual time techniques. Though initially a queue theoretical result, [Parekh and Gallager, 2007] is commonly cited in real-time scheduling papers as well. For example, the virtual time concept is used in the current Linux scheduler, which is described in detail below.

Hierarchical Scheduling Structures

One desirable property in consumer grade systems is to be able to mix real-time applications with regular applications. Often this leads to a construction with a hierarchy of schedulers, typically with some hard-real time scheduler on top and soft real-time and regular best-effort schedulers underneath. [Xingang et al., 1996] suggests using a tree structure where each node is either a scheduler node or a leaf node. Parents schedule their children until leaf level, where the regular tasks sit, is reached. An example is provided in Figure 3.1. The paper also describes a variant of WFQ called Start-time Fair Queuing (SFQ) that provides better guarantee of fairness if the amount of available processing power fluctuates over time.

The hierarchical approach to scheduling is also proposed by other groups. The RTAI/Xenomai extensions to Linux runs a RT-scheduler as root and the Linux operating system as a thread (see Section 3.7). The structure is similar to the one proposed in [Xingang et al., 1996]. The Bandwidth Server class of RBS algorithms,

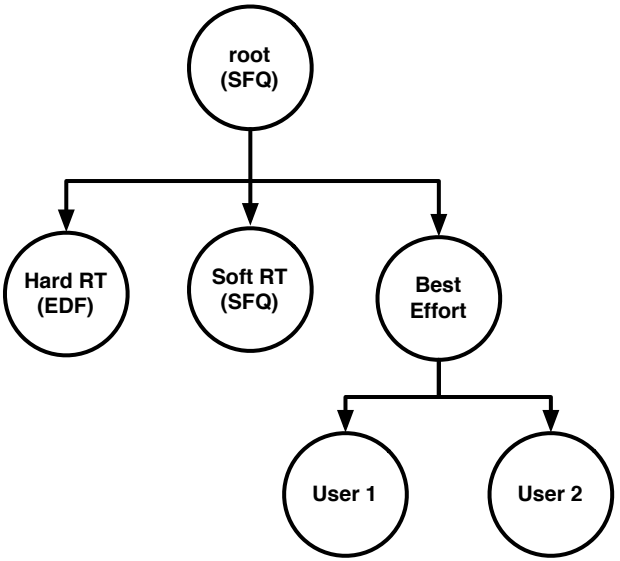


Figure 3.1 Hierarchical structure with schedulers. Note that SFQ is used on more than one level.

detailed in Section 3.2, also uses a hierarchy of schedulers, typically with an Earliest Deadline First (EDF) scheduler [Buttazzo, 1997] on top. Hard real-time tasks are scheduled directly by the EDF algorithm, while soft real-time tasks have dedicated "servers" that dynamically set their deadlines to achieve CPU reservations. Regular applications can be scheduled by a separate server. Lipari et al presented a hierarchical Constant Bandwidth Server construct called the H-CBS in [Lipari and Baruah, 2001] in 2001.

The choice of top level schedulers becomes more critical in the case of insufficient resources. Fixed priority schedulers favor high priority tasks while EDF schedulers will spread out the effects [Cervin et al., 2002] [Buttazzo et al., 1995].

Bandwidth Servers

The concept of bandwidth servers was derived from Dynamic Priority Servers (DPS) by Buttazzo [Buttazzo, 1997]. DPS is a method to accommodate aperiodic or sporadic tasks in fixed priority systems, essentially through a hierarchy of schedulers. The Priority Server is a periodic task with a specified execution time. Arriving aperiodic tasks are placed in a queue and executed by the Priority Server when it is scheduled to run. In the original formulation, unused capacity is just lost.

Buttazzo brought the concept of a server presiding over a predetermined amount of CPU capacity to the dynamic scheduling algorithms. The Dynamic Priority Ex-

change Server (DPE) and the Total Bandwidth Server (TBS) [Spuri and Buttazzo, 1996] were the first formulations using EDF as a root level scheduler. The objective was still handling aperiodic tasks and a lot of theory concerned handling of unused bandwidth. In 1998, Buttazzo and Abeni published [Abeni and Buttazzo, 1998b], which introduces the Constant Bandwidth Server (CBS). By then, the Continuous Media (CM) problem had already been addressed using the Bandwidth Server metaphor by [Kaneko et al., 1996].

Constant Bandwidth Server

The CBS formulation is a popular construct for software reservations and is explained further in this section.

Consider a set of tasks τ_i where a task consists of a sequence of jobs $J_{i,j}$ with arrival time $r_{i,j}$. Let C_i denote the the worst case execution time (WCET) in the sequence and T_i the minimum arrival interval between jobs. For any job, a deadline $d_{i,j} = r_{i,j+T_i}$ is assigned.

A CBS for the task τ_i can then be defined as:

- A budget, c_s, and a pair (Q_s, T_s) where Q_s is the maximum budget and T_s is the period. The ratio $U_s = Q_s/T_s$ is called the server bandwidth. At each instant, a fixed deadline, $d_{s,k}$, is assigned to the server with $d_{s,0} = 0$.

- The deadline $d_{i,j}$ of $J_{i,j}$ is set to the current server deadline $d_{s,k}$. If the server deadline is recalculated, then so is the job deadline.

- When a job associated with the server executes, c_s is decreased by the same amount.

- When $c_s = 0$ the budget is replenished to the value of Q_s and the deadline is recalculated as $d_{s,k+1} = d_{s,k} + T_s$. This happens immediately when the budget is depleted, the budget cannot be said to be 0 for any finite duration.

- Should $J_{i,j+1}$ arrive before $J_{i,j}$ is finished, it will be put in a FIFO queue.

Variations on the CBS formulation

CBS-hd This reformulation of the replenishment rule makes it possible for very overrun sensitive applications to get access to parts of its budget on a shorter deadline. This is investigated in [Caccamo et al., 2000].

The Control Server (CS) Cervin and Eker presented in [Cervin and Eker, 2003] a modification to the CBS scheme that would make it easier to handle the timing needs of a control application. The server budget c_s is here spread out over a number of smaller segments, reducing the uncertainty as to when an input will be read, an output be set, or a code function executed.

Fair Queueing

Fairness was originally introduced by Nagle in [Nagle, 1987] using an informal definition saying simply that a fair algorithm divides the resources between peers equally. The paper also includes what is essentially a prototype of the WFQ algorithm but with little formalism. [Demers et al., 1989] builds on [Nagle, 1987], providing formal definitions and analysis. An algorithm for dividing a resource is defined as fair if

- no user receives more than its request,
- no other allocation scheme satisfying the first condition has a higher minimum allocation and
- the second condition remains recursively true as we remove the minimal user and reduce the total resource accordingly

For applications, the conditions can be expressed in another way. Assume the existence of a finite resource D and n users of that resource. Each user "deserves" a fair share equal to D/n of this resource, but is allowed to ask for less, in which case the difference can be allocated to a user who would like more. Let d_i denote the share a user requests and a_i the share he is given. The maximally fair allocation is then defined so that the share d_f is computed subject to the following two constraints:

$$\sum_{i=1}^{n} a_i = D \tag{3.1}$$

$$a_i = min(d_i, d_f) \tag{3.2}$$

To quantify the fairness of an measured allocation $\{a_1, a_2, ...\}$ compared to the maximally fair allocation $\{A_1^*, A_2^*, ...\}$[Jain et al., 1984] proposes to use a fairness function

$$Fairness = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{3.3}$$

usually referred to as Jain's Fairness Index, where $x_i = a_i/A_i^*$. The fairness will be between 0 and 1, where 1 represents a maximally fair allocation.

Using this metric, we can discuss how fair an algorithm is, how quickly it achieves it, and how sensitive it is to fluctuating conditions. More notions of fairness does, however, exist. The formulation above is limited to calculating the overall fairness, but is difficult to apply to specified time intervals. For that, we need a more advanced formulation. In [Golestani, 1994], Golestani introduces a notion of fairness based on the concept of normalized service. Let r_i be the service share allocated to a task τ_i and $W_i(t)$ the aggregate amount of service this task has received in the interval $[0,t)$. The normalized service is then $w_i(t) = \frac{1}{r_i} W_i(t)$. An algorithm is then considered fair in an interval $[t_1, t_2]$ if

$$w_i(t_2) - w_i(t_1) = w_j(t_2) - w_j(t_1) \tag{3.4}$$

or, in a more compact notation,

$$\Delta w_i(t_1,t_2) - \Delta w_j(t_1,t_2) = 0 \tag{3.5}$$

for any two tasks τ_i and τ_j that have enough work to execute during the entire interval and fair if this is true for any interval.

Unless work is infinitely divisible and all tasks can be serviced simultaneously, all scheduling algorithms relying on resource multiplexing will be unfair if $t_2 - t_1$ is chosen sufficiently small. The theoretical case that allows $t_2 - t_1$ to go towards 0 is called *fluid resource sharing* and is discussed in [Parekh and Gallager, 2007], that analyzes the Generalized Processor Sharing algorithm (GPS). Note that GPS would be completely fair given both definitions of fairness.

Variations of WFQ

While simple in concept, WFQ suffers from being computationally expensive and sensitive to fluctuating resource availability. Several alterations to the original algorithm have been proposed to reduce these problems. For instance, the Start-time Fair Queueing approach mentioned earlier was introduced in [Xingang et al., 1996] as one way of increasing robustness to resource fluctuations, while the Self-clocked Fair Queuing scheme [Golestani, 1994] removes the need to explicitly calculate the ideal processor sharing solution.

The Completely Fair Scheduler The Completely Fair Scheduler (CFS) is a Linux scheduler that was introduced by Ingo Molnar in the 2.6.23 release of the kernel. The scheduler is called "The Completely Fair Scheduler", but the design document recognizes that absolute fairness is impossible on actual hardware. The scheduling principle is simple, each task is given a `wait_runtime` value which represents how much time the task needs to run in order to catch up with its fair share of the CPU. The scheduler then picks the task with the largest `wait_runtime` value. On an system with Fairness Index 1, `wait_runtime` would always be 0.

The implementation of this is slightly less simple. Each CPU maintains a `fair_clock` which tracks how much time a task would have fairly got had it been running that time. This is used to timestamp the tasks and then to sort them, using a red-black binary tree, by the key `fair_clock - wait_runtime`. Multicore scheduling is supported though a partitioned scheduling scheme, with penalties for migrating to other CPUs. Weights are also used, but as is common in POSIX systems they are called nice levels and have the reverse meaning (a nice process would have a low weight). `wait_runtime` is also constrained so that heavy sleepers will not lag too far behind.

In subsequent patches, the group scheduling framework was introduced. In short, it is a hierarchical scheduling scheme where the run queue can be made up of both individual tasks and groups of tasks. The initial intent was to allow fair sharing of the CPU between users rather than tasks. However, the introduction of control

groups (see below) made it possible to do arbitrary groupings, thereby making it a simple but flexible tool for CPU reservations.

At the time the CFS was being merged into the kernel mainline, there were several competing initiatives to bring reservations to the Linux scheduler. The winning patch-set introduced control groups, a general system for grouping tasks and annotating the groups with parameters. These parameters could then be used by various kernel subsystems without the need to change the POSIX task model. It is important to note that control groups in themselves do not alter the behavior of the system, they are just an organizational tool. It is up to the respective subsystems to then interpret the parameters. Some examples of control group-aware subsystems are

- the CFS Group scheduler,
- the CPU affinity subsystem ("cpusets"),
- the group freezer (suspends all tasks in the group) and
- resource accounting.

Adding new control group-aware subsystems is at the time of writing the preferred way to introduce new user controllable functionality in the kernel, instead of adding new system calls.

Comparison between CBS and FQ

Having introduced both the bandwidth server and fair queuing approach to RBS, we can now compare the two methods and see how they differ. Such a comparison is presented in [Abeni et al., 1999], which is summarized here.

First we take a look at the interface they provide for reserving bandwidth. The CBS dedicates an absolute share while FQ uses relative shares. FQ can emulate CBS but with the need to dynamically recalculate the weights when a new task is admitted. FQ algorithms also typically provide bounds on delay, which can be seen as a bound on what deadline requirements a new task can pose. Both schemes have been extended with feedback to adjust weights or bandwidth to achieve some QoS set-point. On the other hand, FQ can more easily be used with mixed real-time and non-real-time tasks.

The run-time properties of the algorithms are also different. CBS does not use quantified time which makes its performance more consistent over varying hardware platforms. FQ is on the other hand simpler to implement. FQ enforces fairness at all times while CBS only guarantees bandwidth allocations between deadlines, making it less conservative. The paper makes the case that FQ is not suitable for media applications as it lacks the notion of task period or deadline, but one can argue that the maximum lag property of an FQ algorithm is a global deadline guarantee, shared by all tasks currently in the system. It is, however, true that the maximum lag

often depends on the number of tasks in the system and the distribution of weights, making the temporal isolation property of FQ weaker. The paper also states that FQ would generate many context switches in order to enforce fairness. While CBS will have context switches as a function of the smallest period server, FQ uses a fixed scheduler time quanta for all tasks. However, as seen with, e.g., the Completely Fair Scheduler, scheduling allowances can be worked in to reduce the number of context switches, at the cost of worse moment-by-moment fairness.

Latency-Rate Servers

In [Stiliadis and Varma, 1998], a generalization of different FQ algorithms are proposed. The class of schedulers called Latency Rate servers (LR-servers) are defined as any scheduling algorithm that guarantees that an average rate of service offered to a busy task, over every interval starting at time Θ from the beginning of the busy period, is at least equal to its reserved rate. Θ is called the latency of the server. A large set of the FQ algorithms fit into this class, including WRR, WFQ and SCFQ. Even non-fair algorithms can qualify (one such example presented in the paper is the Virtual Clock algorithm [Zhang, 1990]).

[Stiliadis and Varma, 1998] goes on to derive a number of results for this rather general class of schedulers, including delay guarantees and fairness bounds. One interesting result is that a net of LR-servers can be analyzed using one equivalent single LR-server. This can be useful when considering a hierarchy of schedulers.

Alpha-Delta abstraction

Similar to the LR-server formulation is the Alpha-Delta abstraction proposed in [Mok et al., 2001b]. Bounds for minimum service α and delay Δ, corresponding to rate and latency in the LR-server formulation respectively, are here derived from a real-time scheduling point of view.

3.3 Feedback allocation control

Adaptive Reservations

One problem when doing RB scheduling is that the execution time for a periodic task may vary over time. As it is undesirable to base our calculations on the worst case, it is likely some deadlines will be missed. While the CBS scheme can handle transient overruns, non transient changes will lead to eventually infinite deadlines (instability). One way to remedy this would be to dynamically set the budget for a server based on prior overrun statistics in a feedback control manner, though commonly referred to as adaptivity in computer science publications.

In [Abeni and Buttazzo, 1999], Abeni and Buttazzo introduce a metric called the scheduling error. If we have a periodic task τ_i with period T_i, then the scheduling error ε_s is defined as

$$\varepsilon_s = d_s - (r_{i,j} + T_i), \tag{3.6}$$

i.e., the difference between the server deadline and the soft deadline of the task. Feedback using the server budget Q_s as the control signal would then be used to drive ε_s towards 0.

A few design techniques for such a controller are discussed in [Marzario et al., 2004]. For the purpose of making the analysis simpler, a restriction is imposed so that even if there is extra unused bandwidth available, a task τ_i scheduled by a CBS will only receive the bandwidth Q_s, a so called *hard reservation*. Assume that τ_i is a periodic task being served with a CBS, divided into jobs $J_{i,j}$ with the corresponding release times $r_{i,j}$. This gives $r_{i,j+1} = r_{i,j} + T_i$, where T_i is the task period. Each job is associated with a soft deadline $d_{i,j} = r_{i,j} + T_i$, that is $d_{i,j} = r_{i,j+1}$. Let $f_{i,j}$ be the actual finish time for $J_{i,j}$ and $v_{i,j}$ be the finish time had τ_i been running alone on a CPU with the fraction $b_i = Q_s/T_s$ of the actual CPU speed, i.e., the virtual finish time. The article uses a modified definition of the scheduling error compared to 3.6

$$\varepsilon_{i,j} = (f_{i,j-1} - d_{i,j-1})/T_i \tag{3.7}$$

which is the scheduling error experienced for $J_{i,j-1}$. Note that since hard reservations is being used, having both $\varepsilon_j > 0$ and $\varepsilon_j < 0$ is undesirable since the task would either be missing deadlines or wasting bandwidth. The relation

$$v_{i,j} - \delta \leq f_{i,j} \leq v_{i,j} + \delta \tag{3.8}$$

where $\delta = (1 - b_i)T_s$, tells us that we can make the CBS approximate the General Processor Sharing (GPS) algorithm by letting T_s go towards 0. Even with normal choices of T_s it is reasonable to use 3.7 and 3.8 to approximate the scheduling error with

$$e_{i,j} = (v_{i,j-1} - d_{i,j-1})/T_i \tag{3.9}$$

A difference equation for the evolution of the scheduler error is then presented in [Abeni et al., 2002]. The paper also proposes a predictor based control structure and three examples of control design using invariant based design, stochastic dead bead design and optimal cost design respectively.

Real-Rate Scheduling

One of the first examples of rate-based scheduling was proposed in [Goel et al., 2004]. The novel approach is to use some task output to measure the rate of progress and thereby eliminate the need for the software designer to assign deadlines or CPU share directly. Experiments presented in the paper are performed using a slightly modified Linux 2.0 series kernel augmented with an rate monotonic scheduling-based RBS scheme. A task with no known period or CPU share requirements but a measurable progress is in [Goel et al., 2004] called a real-rate task.

The example studied is a video pipeline with a producer and a consumer that exchange data via a queue. Queue fill level is the metric used for progress. The scheduler samples the queue and decides if either of the two is falling behind or

getting too far ahead. They use a half-filled queue as the set-point and then design a PID controller to decide the CPU share needed. The period is decided using an heuristic based on the size of the share, lower share meaning longer period.

Heartbeat

Rate-based allocation is further supported by the Heartbeat framework [Hoffmann et al., 2010], which proposes instrumentation of applications so that significant events can be detected by the resource reservation framework. The framework provides an API for registering applications with the resource manager and specifying rate and latency set points. Rates are estimated from events using sliding window methods but individual time stamps of events are saved in order to facilitate more in-depth analysis.

A Linux-based implementation is presented together with results from experiments on a number of application types, including image analysis and video playback.

3.4 Allocation

With the establishment of a variety of RBS techniques, the next important question to discuss is how to calculate the reservations. Given a set of timing sensitive applications, individual application performance can be sacrificed to obtain better global performance. The theory of splitting a resource between consumers is often called resource allocation, though considering its mathematical properties, constrained control would be just as accurate.

Within the field of operations research, using optimization to solve logistics and resource allocation problems is common practice. Some of the iconic problems have been formulated here, including the knapsack and bin packing problems. Solving knapsack- and bin packing problems exactly is of NP-complete complexity [Kellerer et al., 2004; Coffman et al., 1997], making them unattractive for on-line use in limited computational capacity settings.

Constrained control theory

A popular tool for managing constrained dynamics in control is the Model Predictive Control (MPC) formulation. The default setup is postulating a convex cost function of the state trajectories, using an LTI-model as trajectory constraints [Maciejowski, 2002]. Though mathematically feasible to use for allocation problems, solving convex optimization problems can be very resource demanding if special care is not taken.

The Explicit MPC formulation uses pre-calculated solutions, thereby converting computational complexity to a need for storage space. As a result, optimization problems typical for MPC can be solved in milliseconds [Zeilinger et al., 2009; Geyer, 2005]. Calculations can be sped up even further using hardware-based

solvers generated from the problem description, as shown in [Jerez et al., 2012]. The assumption that problem structure is known and fixed does, however, make pre-computed solutions impractical for dynamic resource allocation.

Another way of obtaining very fast and efficient solvers is to generate specialized solvers based on the problem formulation. The CVXGEN package is specifically targeted at embedded platforms, but does not support fixed point arithmetics and is limited to linear- and quadratic programming problems [Grant and Boyd, 2010; Mattingley and Boyd, 2012].

Parallelizing solvers and making it possible to obtain at least partial results before the optimization has terminated are other approaches that can be useful when solving optimization problems in real time. [Giselsson et al., 2013] presents a distributed gradient-based algorithm that uses short iterations together with convergence rate results.

Q-RAM

In 1997, R. Rajkumar presented his Quality-of-Service-based resource allocation model, Q-RAM [Rajkumar et al., 1997]. In essence, this states the allocation as a single objective constrained optimization problem. The model as it was introduced allowed for multiple tasks using multiple resources. Tasks are given utility functions based on the allocated resources, but no technique for how to model a specific task is presented. Neither is the problem of solving constrained optimization problems online discussed. In a later paper [Saewong and Rajkumar, 1999], Rajkumar suggests one way of overcoming the NP-hard problem of general multi-resource allocation, but neither this paper discusses the algorithmic properties of the problem in detail.

3.5 Reservation frameworks

OCERA

OCERA [*OCERA* 2010] stands for Open Components for Embedded Real-time Applications, and was a European project, based on Open Source, which provided an integrated execution environment for embedded real-time applications. From a RBS point of view, OCERA offers a number of interesting components. The OCERA code is based on the RTLinux extension. The patches are applicable to Linux kernels up to version 2.4.18.

Scheduler Patch OCERA modified the Linux kernel so that it provided "hooks" for modules implementing generic scheduling policies. The patch used for this is called the Generic Scheduler Patch (GSP), intended to modularize the scheduler part of the kernel so that additional scheduler policies could be added without further modification to the kernel source.

Integration Patch The Preemptive Kernel Patch is made to work with RTLinux using OCERA's Integration Patch. The Preemptive Kernel work was done by Robert

Love with the aim of improving latency by making system calls possible to preempt [Rostedt and Hart, 2007].

Resource Reservations Scheduling module A dynamically loadable kernel module that provides a resource reservation scheduler for soft real-time tasks in user space is distributed with the OCERA components. It uses a CBS-based algorithm, modified to handle some practical issues. It includes optional slack reclamation functionality using the GRUB algorithm. The module provides a new scheduling policy, `SCHED_CBS`.

Quality of Service Manager OCERA also provides a QoS management services module. This is more or less a controller which changes the bandwidths according to the scheduling error. The approach is more or less that presented in Section 3.3.

AQuoSA

AQuoSA [*AQuoSA* 2010] stands for Adaptive Quality of Service Architecture and is another initiative to bring QoS to the Linux kernel. It builds on the work provided by OCERA and was partially sponsored by the European FRESCOR project. Structure-wise AQuoSA retains the components used by OCERA. AQuoSA has adopted the IRMOS scheduler, a hierarchical EDF-based scheduler described further in Section 3.5.

FRESCOR

FRESCOR [Harbour, 2008] (Framework for Real-time Embedded Systems based on COntRacts) focuses on hard-realtime and contract based resource management. The project background is similar to that of this thesis, dealing with the problem of unknown execution times, mixed requirements and maximizing resource utilization.

The idea is to automate much of the real-time analysis by exposing an API in the form of service contracts. Once the user has specified the requirements, the platform negotiates all current contracts to see if there is a valid solution.

A contract is a set of attributes describing the resource needs of the application as well as certain application properties. Some central concepts here are

- resource type (e.g., process, network or memory),
- minimum budget (WCET/T),
- maximum period, and
- deadline.

These parameters are then used for automated response time analysis to determine if the resource requirements can be met.

Spare capacity distribution In order to address the goal of maximum resource utilization, FRESCOR suggests a form of slack reclamation here called Spare capacity distribution (SCD). As the admission policy works under worst case assumptions, it is likely the system will have spare capacity in most situations.

ACTORS

ACTORS (Adaptivity and Control of Resources in Embedded Systems) was a European Union funded research project with the goal to address design of resource-constrained software-intensive embedded systems. The strategies employed included

- data flow-based programming and code generation,
- virtualization and
- feedback resource management.

One of the primary deliverables from the project was an implementation of an adaptive resource management framework [Bini et al., 2011].

ACTORS-model In [Segovia and Årzén, 2010] the authors present a model, using a set of discrete resource consumption and quality output levels. The problem formulation concerns both assigning tasks to cores and dimensioning resource reservations for each core. The domain mentioned as the target in the paper is data-flow applications, but there is nothing explicitly in the model that ties it to this.

One drawback of this approach is the need to supply the resource levels. This is non-trivial and increases the complexity of developing applications. The approach taken in the paper requires only one defined level and utilizes a continuous quality measure, thereby making the optimization easier to solve.

Figure 3.2 shows an architectural overview of the actors resource management model, where a physical platform is virtualized and presented as a number of resource constrained virtual platforms to the applications layer. The architecture supports mapping one CPU core into many virtual cores, but while multicore systems are supported the model does not allow for a virtual resource to stretch over many physical.

The resource manager (RM) allocates virtual platforms to applications as they request resources and monitor the application performance in runtime in order to adapt resource levels for optimum system performance. Each RM aware application is designed with a number of predetermined Service Levels (SL), each corresponding to a nominal resource demand and performance.

The problem of assigning virtual platforms to actual hardware and selecting service levels for all running applications is posed as a mixed integer programming problem and solved using the GNU Linear Programming Kit [*GLPK*]. This was

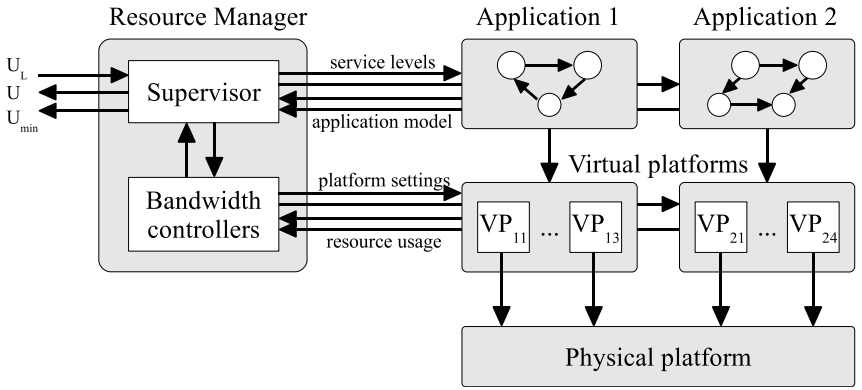


Figure 3.2 The ACTORS resource architecture.

later restated as a game theoretical mechanism design problem where the applications are allowed to select their own service levels, enabling a more distributed approach [Maggio et al., 2013].

Reservations and online adaptation

The ACTORS RM utilizes CBS style reservations through the SCHED_EDF patch set for the Linux kernel, as described in Section 3.5. Once the initial resource assignments are done as part of the global optimization described above, actual resource usage is monitored at runtime. Overrun statistics provided by the EDF-scheduler as well as resource consumption information provided by the process accounting system are the basis for increasing or decreasing allocated resources, as described in [Romero Segovia et al., 2010].

SCHED_EDF / SCHED_DEADLINE

A significant deliverable from the ACTORS project is an EDF-scheduler implementation for Linux kernels 2.6.30 and later. It is specifically designed for resource virtualization and introduces hard CBS type RBS as the top tier scheduler. Notably, it has support for multicore platforms.

As of kernel version 3.14-rc1, the SCHED_DEADLINE patch set was merged into the main line tree for official support. The patch adds

- a top tier EDF scheduler,
- bandwidth, deadline and period as new process parameters, and
- new system calls (sched_setattr/sched_getattr) through which these parameters can be manipulated.

The implementation does have multicore support, but the official documentation included with the initial release states that some issues around task migration still remains to be resolved.

The inclusion of a true EDF scheduler and support for CBS style reservations in the Linux kernel can mean a great deal to wide spread adoption of reservation based scheduling in consumer products and general computer systems, as such functionality has historically only been available in research grade software or expensive proprietary operating systems.

IRMOS Scheduler

The IRMOS scheduler [Cucinotta et al., 2011], developed by the EU funded project Interactive Realtime Multimedia Applications on Service Oriented Infrastructures, is a hierarchical scheduler patch for the Linux kernel with many similarities to SCHED_DEADLINE, but with two notable features. First of all, it is a hierarchical scheduler that permits other schedulers to be run inside it. Secondly, while SCHED_DEADLINE is a partitioned EDF scheduler, IRMOS has an experimental global EDF implementation.

3.6 The Xen hypervisor

Virtualization technologies make it possible to partition a physical computer into several logical instances, each one running a separate operating system that experiences itself as running alone on the hardware. The layer beneath the OS layer is sometimes called the hypervisor layer as it uses a special mode enabled in modern CPUs called the hypervisor mode. Xen is an open source hypervisor created in 2003 at the University of Cambridge [Barham et al., 2003].

Architecture

A Xen system has multiple layers, the lowest and most privileged of which is Xen itself. Xen can host multiple guest operating systems, each of which is executed within a virtual instance of the physical machine, a domain. Domains are scheduled by Xen and each guest OS manages its own applications. This makes up a hierarchy of schedulers with the Xen scheduler on top.

Xen supports several top level schedulers, including EDF and WFQ. This makes it possible to apply resource reservation to the virtual machines running on top of Xen.

3.7 Xenomai

Xenomai [*Xenomai* 2010] is a real-time development framework cooperating with the Linux kernel, in order to provide a pervasive, interface-agnostic, hard real-time

support to user-space applications, seamlessly integrated into the GNU/Linux environment. It is an alternative to RTLinux and RTAI and is a possible platform for developing RBS schemes. It was launched in 2001 and in 2003 merged with the RTAI project. The projects split again in 2005, going after separate goals.

Xenomai achieves superior real-time performance compared to standard Linux while still allowing regular applications. The real-time kernel is a small, efficient run-time which executes the Linux kernel as a low priority task. Real-time tasks will then be run by the RT kernel, next to the Linux kernel. The Xenomai kernel also provides an API with extensive support for real-time primitives that the Linux kernel lack, including support for periodic task models. Xenomai uses a hypervisor similar to the Xen hypervisor, but focuses on real-time performance. For example, it allows real-time tasks to receive interrupts even if a Linux process has requested interrupts to be turned off.

3.8 Linux Control Groups

The introduction of Control Groups in the Linux kernel provided an extensible framework for hierarchical organization of processes. The patch came as a response to a series of patch sets aiming to add resource reservation functionality to the Linux kernel. Rather than adding more subsystems, the control group patch implemented the support by extending and generalizing the existing cpusets functionality. Control groups extracts the process grouping code from the former cpusets implementation into a generic container system, and then reimplements cpusets on top of that as one of many possible modules. The intention is that the various resource management and virtualization efforts can also become modules rather than separate patch sets, with the result that

- the user space APIs are (somewhat) normalized
- the additional kernel footprint of any of the competing resource management systems is substantially reduced, since it does not need to provide process grouping/containment, hence improving their chances of getting into the kernel

Together with the aforementioned group scheduling mechanism introduced in Linux 2.6.24 (see Section 3.2), control groups enables a primitive form of resource reservations for processes and users.

3.9 Estimating software model parameters

Uncertainty and time varying dynamics can be handled through on-line estimation techniques, as is common in feedback and adaptive control. For a system with vary-

ing time scales, both over time and between components, traditional periodic sampling will not fit well.

Recently, event based control and estimation has gained attention. Such theory is attractive because it can be more efficient than periodic sampling and also better account for situations where information is delivered in an event based fashion. This thesis has been inspired by such works as [Henningsson and Cervin, 2009; Sandee et al., 2007].

4

Power and energy management

This chapter introduces prior research related to power management and energy management for cyber-physical systems (CPS). The chapter starts with an introduction of the problems associated with power management in mobile systems and how they relate to the resource allocation problems presented in the previous chapter. After that follows research related to minimizing power consumption through scheduling techniques, both for the single core system and computer clusters. Finally some results related to the explicit management of heat dissipation and power consumption are presented.

4.1 Energy and resources

In previous chapters, the need for resource management techniques for situations with time varying resource availability was discussed at length. In mobile systems, the constant attention to energy concerns is perhaps the most important source of resource variability. Tightly coupled with this is heat management, as all forms of energy expenditure invariably leads to the development of heat, the dissipation of which is a problem in all high performance computer environments, mobile or not.

Thermal RC-models

Given the limitations on how a mobile device can dissipate heat, having appropriate models for describing temperature dynamics within the system is necessary for proper power management. Such models must not only consider how heat is produced through the operation of the system components, but also how it propagates between them. Controlling peak temperatures and temperature gradients thus become part of the resource management problem.

A popular model structure used in many papers likens the flow of thermal energy to that of electric currents in a circuit made up of resistances and capacitances. In

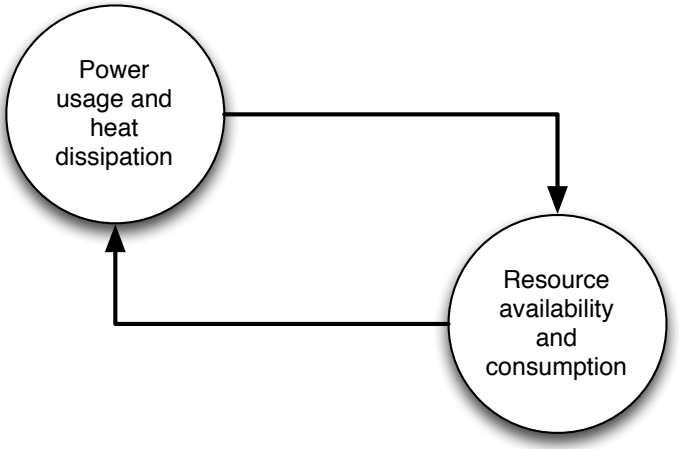


Figure 4.1 In the cyber-physical world, physical and computational dynamics affect each other, making it necessary to develop models that can encompass both aspects.

these models, temperature fills the role of potential from the electric counterpart, whereby thermal energy flows from high temperature nodes to low temperature nodes, limited by the conductive properties of the connecting medium, modeled as resistances.

The simplest possible model would connect one node with the surrounding medium. If the temperature in the node is denoted T, the surrounding temperature denoted T_o and the thermal resistance between them denoted R, the resulting heat transfer from the node to the surrounding environment H can be calculated as

$$RH = T_o - T \tag{4.1}$$

The effect of a flow H on a node with temperature T is described through the physical property of thermal capacitance, here denoted C, and obeys the equation

$$C\dot{T} = H. \tag{4.2}$$

Through the application of the thermal equivalents of Ohm's Laws and Equations 4.1 and 4.2, larger structures of thermal RC-circuits can be analyzed, as exemplified in [Skadron et al., 2002; Ferreira et al., 2007b]. Though the physical parameters are not easily derived analytically from known system parameters, the linear structure of thermal RC-models makes it easy to apply techniques of system identification to find the parameter values.

Adding active elements that consume energy and produce heat consists primarily of adding more heat flows. Commonly the CPU is considered to be described

by two power levels: an idle level and a max load level, with a linear interpolation model for loads between 0% and 100%, or formally

$$P_{CPU} = P_{0\%} + U(P_{100\%} - P_{0\%}) \tag{4.3}$$

where U denotes the load.

[Skadron et al., 2002] marks one of the earliest applications of these models for heat management in computer systems. Over time the thermal RC-models have seen much refinement, resulting in advanced simulation packages, such as HotSpot [Skadron et al., 2003].

Thermal control techniques

Controlling temperature in relevant nodes of the system can be done through manipulation of the entities in the above equations. The techniques for this fall into mainly two categories, controlling the power used by the system through Dynamic Voltage- and Frequency Scaling (DVFS) or by restricting the load through scheduling policies or as part of resource management.

Clock gating Clock Gating, sometimes referred to as Global Clock Gating or Stop-Go, is a technique for preventing overheating in circuits by stalling the clock signal when the temperature in the chip rises over a threshold level. This will stop all state changes in latches in the circuit, thereby immediately reducing the amount of heat generated. However, as heat can also come from leak currents and external sources, this is not always sufficient to prevent the circuit from overheating and taking permanent damage.

Dynamic Voltage- and Frequency Scaling (DVFS) Intel Corporation introduced its Geyserville (later trademarked as SpeedStep) technology in 1999, bringing variable speed processors into the mainstream, enabling a significant gain in battery life for laptop computers. Since then, the inclusion of DVFS features has been adopted by most CPU manufacturers, although under different names and implementations.

In general, flipping latches in the CPU consumes power, so reducing the frequency with which this happens reduces the power consumption. As the voltage required to power a chip depends on the clock frequency it runs at, reducing frequency also enables reduced voltage and as such the techniques are commonly used together [Aydin et al., 2004].

Power aware scheduling The simultaneous treatment of selecting operating voltage and frequency for a CPU and a schedule for the real-time tasks executing on it, in order to minimize the peak power or energy used by the system, has been treated in a multitude of publications since the introduction of the technology. Starting with methods relying on off line analysis [Aydin et al., 2001], the trend has since been steadily towards making more and more decisions online [Aydin et al., 2004].

Extensions to multicore computing has since been developed [Bautista et al., 2008] as well as for distributed systems, such as data centers [Laszewski et al., 2009; Sharma et al., 2003].

Controlling the power/performance trade-off

As more dynamic techniques are being developed, it has become natural to consider the trade-off between power consumption and system performance or Quality of Service (QoS). In [Sharma et al., 2003], the objective is to minimize the energy consumption while not violating performance constraints. The metric used here is computational delay and the constraint is that all tasks admitted to the system should meet their deadlines, i.e., a hard real-time formulation. In essence, the trade-off decision here becomes if to admit tasks to the system or not.

Other works rely more on user oriented metrics, such as the transaction delay experienced by the user of the system [Chen et al., 2008]. As these metrics are not formulated in terms of hard deadlines, they are soft real-time systems, something which has become a staple of power aware computing systems. The fact that the media applications, typically considered as being soft real-time, stand for a significant portion of the resource demand, has eased this transition.

With this increased focus on measurable performance rather than the certification of hard real-time schedulability has come a noticeable change in paradigm. Using the techniques available through reservation based scheduling, managing the limited resources and the side effects of their consumption becomes more like traditional process control problems, as discussed in for example [Fu et al., 2010a].

4.2 Spatial resource management, "E-logistics"

The theory of real-time systems traditionally concerns the temporal aspects of resource management, answering question predominantly concerning "when" to spend resources (and on what). With the introduction of multiprocessor and multicore computing, the notion of locality, "where", is introduced, i.e., the spatial aspects of the problem. As systems spread out even further, connecting components through system buses or network protocols, spatial effects grow stronger.

Distributed systems can arise by two primary mechanisms, decomposition of monolithic systems or composition of individual parts. What separates a part in a distributed system from another does not have to be space, rather two parts can be considered separate if giving a resource to one does not mean it is immediately available to the other, given some relevant time scale.

Systems on static topologies

A static topology system is one where the structure is explicitly part of the system design. The robot example in Section 2.2 is one such system, where the connections

between components are represented by physical connections. Other examples include task graphs, networked control systems, and queue systems.

Task graph scheduling For system of purely software components, a popular and widely researched formulation is that of task graph scheduling. The objective is to map a set of tasks with precedence constraints onto a set of identical processors, a problem proven to be NP-complete [Robert, 2011]. This formulation has resulted in numerous heuristic designs, most of them static in nature, which makes them infeasible for scenarios with high variability in resource availability. Recent work has moved towards more dynamic formulations, acknowledging the need to consider more uncertain situations [Choudhury et al., 2012].

Networked control systems A common type of static topology system is networked control systems, i.e., where controllers, actuators and sensors are physically separate nodes in a computer network. Since control system performance is strongly dependent on timely measurements and actions, managing limited bandwidth is key to a successful design. Applying control theory to the joint problem of managing the resources and the plant is a natural step, studied by, e.g., [Branicky et al., 2002]. This would later give rise to the event based control structures discussed by, e.g., [Tabuada, 2007; Cervin and Henningsson, 2008].

Queue systems Given the similarities between scheduling of processor time and queue theory as discussed in Chapter 3, it is not surprising to find static topology resource management problems arising in network engineering and queue systems theory, though usually referred to as "congestion control", see for example [Akyildiz et al., 2001; Katabi et al., 2002]. With the ever increasing demands on cost efficiency, it is also expected that more and more dynamic methods are applied in order to refrain from over-provisioning, as exemplified in [Wang et al., 2010].

Systems on dynamic topologies

A dynamic topology system is one where the structures changes over time, either autonomously or driven by decisions. Ad-hoc networks and overlay networks are typical examples of systems with dynamic topologies, but lately the drive for greener computing has led to the studies of how to turn of network links in order to save energy.

Overlay networks and content distribution Digital media and content distribution are arguably the driving forces behind both technology and legislation relating to the internet. It is becoming clear that the current infrastructure is not designed around the sustained data rates associated with streaming music and video and as such services become increasingly popular, there is a strong demand for technological solutions to reduce congestion and energy consumption.

Peer-to-peer (P2P) and peer-assisted solutions have been used in wired networks to reduce infrastructure load [Kreitz and Niemela, 2010], but as mobile devices are

increasingly the point of media consumption, there is a strong demand for extending these solutions to the mobile networks.

The principal complication in the mobile case is the addition of energy constraints. While traditional P2P has proven effective for cooperative content distribution and offloading congested nodes or links in the network, it is not energy aware and as such a phone risks running out of power by serving content to others before it can benefit from accumulated goodwill.

The study of local cooperative techniques that account for the energy cost of sharing data has recently been done by several independent groups. Common among them is that they leverage the lower energy consumption of short range communication and point out the additional benefit of local schemes in that they offload the long range infrastructure [McNamara et al., 2008].

[Jung et al., 2010] introduces a framework for cooperative sharing, starting with an altruistic scheme and then introduce guard conditions to prevent participants from spending too much energy on serving content. The approach is based on a Markov Decision Process (MDP) formulation, which is studied in detailed in [Cheung et al., 2009], and is validated through an implementation on Android based phones showing substantial energy savings. One issue with the experimental setup is that the clients involved are assumed to be in contact with each other indefinitely. Additionally, the approach does not guarantee that participants will actually benefit from the scheme.

[Jin and Kwok, 2010] considers a streaming scenario using a game theoretical approach. The theoretical framework includes both power and bandwidth considerations, but simplifies the game dynamics by assuming that players without sufficient energy will simply not participate. While the real time aspects introduced through the streaming scenario are relevant, the formulation is very abstract and it is not clear how it could be implemented in reality.

[Yaacoub et al., 2012] also approaches the problem using a game theoretical basis, but explicitly addresses the energy issue and fairness. A large file object is distributed to a set of clients by transmitting parts of it to some nodes over long distance link and then distributed to the rest using local communication. The proposed scheme permits selecting utility in a variety of ways, including greedy but unfair minimization of the total energy used by the system and minimizing the maximum energy spent by any client. The proposed solution considers static populations and uses an off-line calculated solution, making it unsuitable for the more dynamic scenarios studied in this thesis.

5

Modeling and Estimation

This chapter discusses what is a suitable base for allocating resources, ways to model the resource consumption and the resulting utility of a software component, and techniques for estimating these quantities and model parameters online. It also presents a model for how resource availability is limited by CPU thermal dynamics. The chapter is based on the publications [Lindberg, 2009], [Lindberg, 2010b], [Lindberg, 2010a] and [Lindberg and Årzén, 2010a].

5.1 Smartphone model

Returning to the example in Chapter 2, Section 2.1, a typical use case would involve core phone services, such as audio and video telephony, the network stack and some 3rd party applications, such as GPS-based navigation software. These applications share a CPU through the OS scheduler, as per Figure 5.1. The system would also

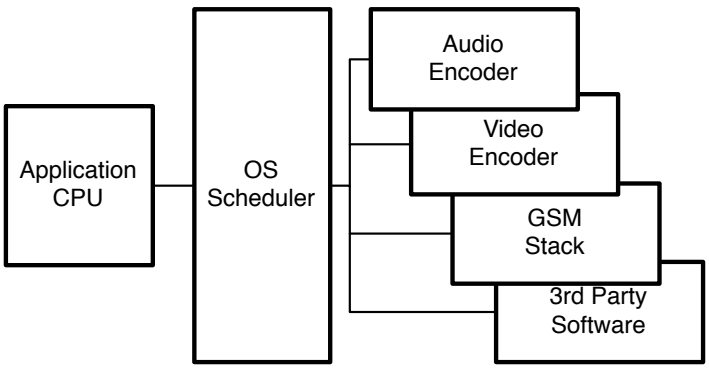


Figure 5.1 A typical use case for a smart phone with support for 3rd party software. The application CPU is commonly a low power chip with limited precision running a fully featured OS such as the Linux-based Android platform.

be running a number of largely dormant system services that will account for a comparatively small part of the total resource needs.

The arrival of Linux-based smart phones has enabled the use of RBS techniques to manage performance, rather than by fixed priority scheduling that is the norm. Managing the CPU resource so that software performance is satisfactory can therefore be seen as an allocation problem.

One complication is that the primary resource, the CPU, is limited in performance by power and heat constraints. Running software on the CPU produces heat and since most cell phone casings do not support active cooling, the CPU temperature must be regulated by limiting usage.

Given the unknown properties of 3rd party components, the number of combinations of enabled software and the varying operational conditions, e.g., ambient temperature, allocation decisions will have to be made based on information collected at runtime.

This chapter will therefore introduce

- a resource consumption and performance model intended for the class of software components expected to be dominant in terms of resource demands,
- a model describing how available CPU resources depend on chip temperature, and
- techniques for estimating model parameters online.

5.2 Allocation and utility

When allocating a limited resource, it is necessary to consider the *utility*, sometimes called Return On Investment (ROI) or reward, gained. The field of Operations Research (OR) is primarily about making such decisions, typically employing constrained optimization.

Assuming that the applications of interest are timing sensitive rather than hard real-time, one approach would be to model the utility of an allocation as a function of the resulting task performance. If hard realtime requirements are assumed, the notion of individual task performance must be dropped; as a task either performs nominally or fails. Deciding what tasks to admit and which to reject based on their nominal performance is essentially the knapsack problem, which unsuitability for use in these contexts has already been discussed in Section 3.4.

Deadline miss ratio

Using the number of task deadlines missed per time interval, Deadline Miss Ratio (DMR), as a measure of utility has been suggested by several previous works, e.g., [He et al., 2007]. This has been primarily used for feedback resource management purposes, but given a model of task execution times, a prediction of the DMR is

feasible to use. The drawback of such a scheme is that it does not explicitly take into account that tasks can adapt to resource availability. This is commonly practiced in modern software, with examples in media processing [Isovic and Fohler, 2004], computer games [Foundry, 2010] and control [Årzén, 1999]. An adaptive task could therefore theoretically have the same utility regardless of resource allocation.

Rate-based utility

For media applications, the quality of the output is strongly connected to the processing rate. This holds true for all parts of the media processing chain, from encoding to decoding. It is therefore natural to consider how resource allocation decisions impact the processing rate of the system and thereby indirectly the Quality of Service. It is possible to view media stream processing as a special case of data flow or stream processing. In these programming models, data is often contained in packets or tokens which are then processed by a network of computational elements. The rate at which data tokens are processed is a very tangible metric for the application performance. Data flow formulations exist for a large group of software relevant to embedded situations, ranging from automatic control to 3D-graphics. This supports making rate an important basis for resource allocation in heterogenous systems.

Compared with DMR, rate-based utility sets an absolute value to the resulting performance, though it cannot account for more subtle effects of adaptive tasks. As an example, a video decoder could decide to drop frames as a mechanism for rate adaptation. Deciding which frames to drop is important for playback quality, as discussed in [Cha et al., 2003]. For now, it will be assumed that if a task has such properties, it is able to make such decisions by itself. Explicitly including such information in the system global model is in this work considered infeasible for a general purpose allocation framework.

Rate is also an application-centric metric which makes it easy to use from a developer's point of view. In many cases the desired execution rate is explicitly decided at design time, as with encoding frame rate for video, simplifying its use even further. Given its central position in this work, the term *rate* deserves a clear definition. *Rate* signifies the number of occurrences of a pre-specified event during a counting time-period.

The choice of event and counting period is strongly situational. Consider for instance the difference between digital audio and video. The ear is much more sensitive to audio jitter than the eye is to frame jitter [Steinmetz, 1996]. The audio stream is also sampled at a significantly higher rate than the typical video stream (16 kHz vs 25 Hz). While loosing one or several movie frames during a second might not even be noticeable for the viewer, loosing the same percentage of audio samples will make the audio sound very distorted. When considering a system with mixed time scales, selecting a suitable time period for allocation can be troublesome. This will be discussed further in Chapters 6 and 7.

Timing-based utility

Another alternative is to use the exact time that a task completes a job as a metric, as opposed to whether it always complete ahead of a deadline. This measure contains more information than the DMR and can be especially useful when co-ordinating dependent components. An example of this is synchronization between audio and video data in a capture device, which is examined in more detail in Chapter 7.

5.3 Components with rate-based utility

This thesis will discuss mostly applications where utility is based on execution rates and cycle timing. The following section discusses how to model single- and multi-resource dependent components. From a resource consumption point of view, there will be no difference made between a single- or a multi-threaded process, or even something made up from a family of processes. The building blocks will be called components, where a component i is an entity that

- consumes resources and outputs results opaquely,
- is responsible for distributing allocated resources to subcomponents,
- has a behavior determined by a dynamic mapping from a set of inputs u_i to a set of outputs y_i, and
- where all significant dependencies affecting performance is modeled through this mapping.

There are two main reasons for this:

- **Composability.** In order to simplify building large systems including 3rd party components, a building block semantic where the set of parts is closed under composition is useful. This thesis will disregard the functional aspects of the component architecture, considering it a mostly solved problem, and focus on the resource-driven temporal qualities.
- **Extending into non-software components.** It is the aim of this work to model systems built from a mix of software and hardware. Therefore, raising the level of abstraction is necessary to hide properties unique to software or hardware parts.

The cyclic component model

The most basic case is a component that depends on a single resource. In a computational setting, this corresponds to the standard problem of a set of independent, CPU-bound tasks. In this work the notions of periods and deadlines are dropped

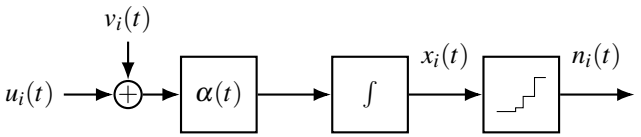


Figure 5.2 A cyclic computation software component. $u_i(t)$ is the CPU resource share assigned, $v_i(t)$ is the disturbance on the assigned share caused by the environment, $\alpha(t)$ is the execution speed of the CPU, $x_i(t)$ is the accumulated execution, the last block is a staircase function mapping $x_i(t)$ onto $n_i(t)$, which signifies the number of completed cycles.

and replaced by the notion of cycles. Furthermore, as the entities of interest can be made up from nested sets of tasks, the term component will be used.

Figure 5.2 shows the logical setup for the *cyclic component model*, used as the template further on. Generally speaking, the component accumulates the incoming resource, in this case executed CPU instructions, until some quantity has been achieved. At this point it outputs the result and commences a new cycle. The details of the model are as follows:

- $u_i(t)$ is the CPU resource share assigned to component i and is a dimensionless number.
- $v_i(t)$ is a zero-mean disturbance of this share caused by the scheduler as experienced by component i. The two major sources for this disturbance is the error in the approximation that the CPU is really fluidly divisible and certain system events that interrupt normal execution, such as hardware interrupts, virtual memory handling or access of shared resources.
- $\alpha(t)$ denotes the speed at which the processor executes instructions and has the unit of completed instructions per time unit.
- $x_i(t)$ is the accumulated number of executed instructions for component i.
- $n_i(t)$ signifies the number of cycles completed by component i at time t and is calculated as

$$n_i(x) = \max k \tag{5.1}$$

$$s.t. \sum_{j=1}^{k} C_i(j) \leq x_i(t) \tag{5.2}$$

for some sequence $C_i(1), C_i(2) \ldots C_i(k)$ that describes the amount of execution it takes component i to complete computation cycle k.

A cyclic component will block if and only if it is starved of CPU-time. The cycle execution time for cycle k is considered to be a weakly stationary stochastic process $C_i(k)$ and h denotes a time interval such that $E\{C_i(k)\} < h$. It is assumed that $C(k)$ has a strictly positive lower bound.

From this it follows that if a component is started at t and executes until $t+h$, the expected number of completed cycles becomes

$$E\{\frac{h}{C_i}\} \approx \frac{h}{E\{C_i\}} \tag{5.3}$$

Assuming the share $u_i(t)$ is constant over h, Equation (5.3) can then be used to approximate the dynamics of $n_i(t)$ as

$$n_i(t+h) - n_i(t) \approx \frac{1}{E\{C_i\}} h u_i(t) = h k_i u_i(t) \tag{5.4}$$

or in words, n_i evolves approximately like a discrete time integrator with the unknown gain k_i and is driven by the input $u_i(t)$. $(n_i(t) - n_i(t-h))/h$ is denoted $y_i(t)$ and is referred to as the *execution rate* of component $\mathscr{C}_i$. Figure 5.3 shows an example of $n_i(t)$ and the corresponding $y_i(t)$ for one of the components used in the simulations. In the figure $h = 1$. As a result, $y_i(t)$ will lag behind by as much. Though the plot does not show it, it is assumed that $n(t) = 0,\ t < 0$, which is the reason for the initial behavior of $y(t)$.

Rate-error utility

While the rate $y_i(t)$ itself can be used as a utility metric, often an algorithm or component is designed with a specific rate in mind. Even if there are resources enough to increase execution rate beyond that, performance would not increase or in some cases even degrade. For this purpose, the component model is augmented by another parameter, the *rate set-point* r_i, denoting the optimal rate for this component. It is common that software components will limit their execution rate once $r_i(t)$ is achieved and the linear mapping between $u_i(t)$ and $y_i(t)$ in (5.4) must be altered to reflect this, resulting in

$$y_i(t) = \begin{cases} k_i u_i(t-h) & 0 \le u_i(t-h) \le \frac{r_i}{k_i}, \\ r_i & u_i(t-h) > \frac{r_i}{k_i} \end{cases} \tag{5.5}$$

Figure 5.4 shows two cases which were produced using MPEG-4 video streams and the free MPlayer software. The videos are encoded at a fixed rate, in this case 30 frames per second (fps). When allocating a lower share of the CPU than required for full rate playback, MPlayer starts to skip frames, thereby adapting to the reduced playback rate.

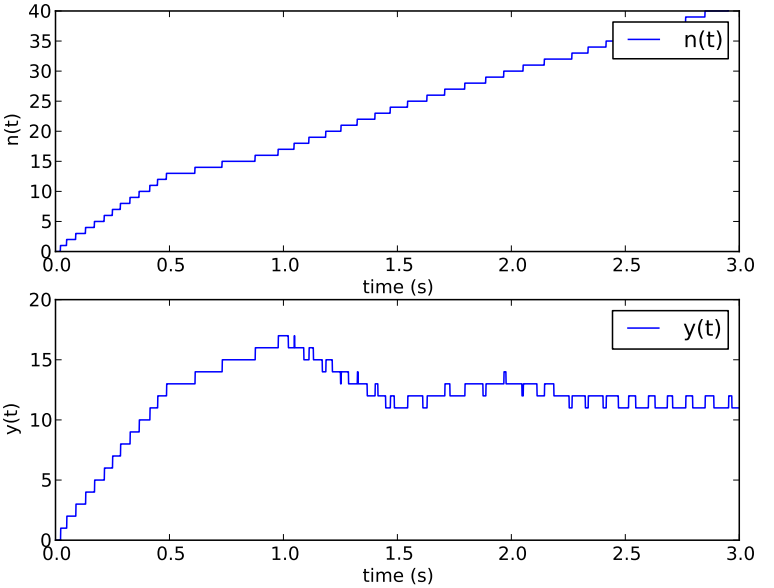


Figure 5.3 $n_i(t)$ and $y_i(t)$ for one of the components in Chapter 7. Note that these are not sampled curves. Despite their jaggedness, both signals are defined for all t. Rate estimation is here done by a one second time window, i.e., $h = 1$. Note that this will make $y_i(t)$ lag behind by as much.

Cycle completion timing

The other aspect of the cyclic component model is the event dynamics, i.e., exactly when the job cycles complete. In the cyclic component model, the completion time for cycle k is denoted $t_i(k)$ and modeled as

$$t_i(k+1) = t_i(k) + \frac{C_i(k)}{u_i(t(k))} \tag{5.6}$$

if $u_i(t(k))$ is assumed to be constant over the interval $[t_i(k), t_i(k+1))$, making it similar to the virtual finish time introduced in [Demers et al., 1989].

While this could be used for traditional deadline driven scheduling (e.g., keeping $t_i(k+1) < t_i(k) + D$)) it is also possible to control other and sometimes more interesting metrics, such as the synchronization between two non-uniform sequences, see Chapter 7.

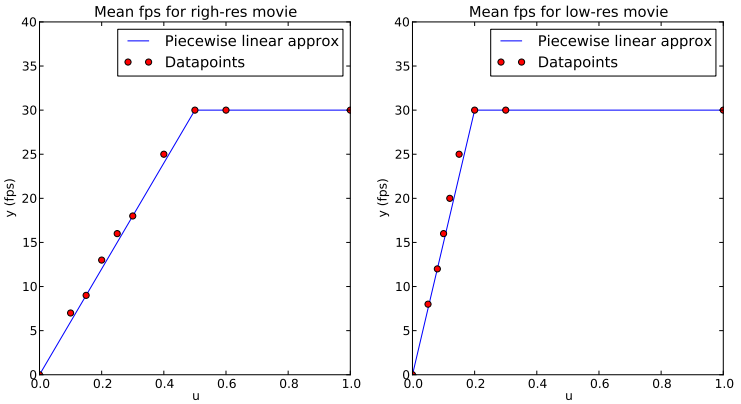


Figure 5.4 Experimental results of controlling CPU-share for the MPlayer decoder using Linux 2.6.27 and Control Groups. The diagrams show how the frame rate per second (fps) depends on the amount of CPU share allocated to the decoder. The rate increases linearly with share until the movie can be played back at encoded rate.

5.4 Multi-resource dependencies

It is desirable to model components that require multiple resources to execute. This thesis proposes to do this by extending the component blocking rule so that a component will block if and only if it is starved of one or more resources. It is assumed that a component has the capability of accumulating incoming flows, e.g., in FIFO queues for data or execution time deficit accounting in the scheduler for CPU-time. It follows that the component execution rate is limited by the rate at which resources are made available to it. Formally, if τ_i is dependent on the resource flows $u_0, ..., u_N$

$$y_i = \min(k_{i,0}u_0, ..., k_{i,N}u_N) \tag{5.7}$$

would describe its execution rate. From (5.7) it follows that an allocation strategy should try to keep the incoming flows equal in order to minimize over-provisioning and reduce the risk of buffer overflow. Furthermore, it points towards two important objectives for maximizing performance of these systems:

1. Calculating a steady state flow that maximizes the relevant performance metric.

2. Control transient effects that cause blocking .

How Objective 2 is connected to the cycle completion dynamics is further discussed in Chapter 7.

5.5 CPU thermal dynamics

In order to take the thermal dynamics of the CPU into account when deciding how much load the system can sustain, this thesis proposes a model for the thermal dynamics based on [Ferreira et al., 2007a]. By limiting the load, or utilization, the temperature can be controlled even if the CPU is only passively cooled. The model of the dynamics from CPU power P to CPU temperature T is on the form

$$\dot{T} = a(T_a - T) + bP + d \tag{5.8}$$

where a and b are constants depending on the thermal resistance and heat capacity of the processor and T_a the ambient temperature. d is a disturbance term which will be assumed to have slow dynamics, such as heat generated by direct sunlight or by being placed on a heated surface. For off-the-shelfs CPUs, a and b are in the order of 10^{-4} and 10^{-3} respectively (see for example [Fu et al., 2010b]), making the dynamics relatively slow. It is therefore assumed that it is possible to filter out measurement noise, which is therefore omitted from the model.

The relationship between CPU load U and P is then modeled as

$$P = P_{0\%} + U(P_{100\%} - P_{0\%}) \tag{5.9}$$

Sampling the combination of Equations (5.8) and (5.9) can then be done under zero-order-hold assumptions.

A step response experiment was carried out on a Pioneer mobile robot [MobileRobots, 2006] with an internal Intel Pentium III-based computer [*Model VSBC-8 Reference manual* 2007] in order to validate the model structure, giving the results shown in Figure 5.5. The on-chip temperature sensor has a sample period of 2 seconds.

5.6 Parameter estimation

A key assumption in this work is that the model parameters are not available beforehand and therefore have to be estimated online. This section discusses the aspects of this and some possible techniques. First the problem of estimating the execution rate from the sequence of observable cycle completion events is discussed. An estimator for the k-parameters based on the rate estimate is then suggested.

The central problem with computational components is that information comes in form of events instead of continuous signals. For example, there is only new information about the execution rate when a calculation cycle completes or when an expected event is missing. There are two main alternatives to estimate the execution rate from this, sliding time window event counting and event based filtering. It is assumed that

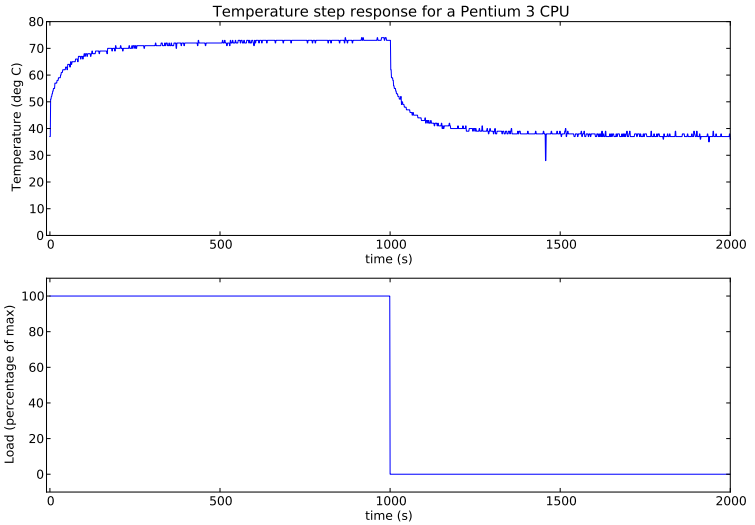


Figure 5.5 Results from a step response experiment performed on a Pioneer mobile robot. The experiments supports a simple first order model for the dynamics between load and chip temperature.

(a) while the variable CPU-speed $\alpha(t)$ cannot be directly measured, it is possible to measure the accumulated execution $x_i(t)$ and thereby

$$x_i(t) = \int_{t_0}^{t_1} \alpha(t)dt$$

(b) The completion of a cycle is observable through an event, defined as the tuple $(t_i(k), x_i(t_i(k)))$, where $t_i(k)$ is the time when component i completed cycle k.

The following properties are considered important for the resulting algorithm

- Time complexity
- Space requirements
- Sensitivity to noise
- How fast it can detect a change in rate

Sliding window event counting

Using the definition of rate (events/time period) it is natural to consider an approach where the number of events occurring over a predetermined time period is mechanically counted. Given a suitable window length, the method is straightforward in implementation, but needs an unknown amount of memory to keep the events, also the time complexity is proportional to the rate, i.e., unknown beforehand.

Event based filtering

An alternative approach is to view the estimation of y_i as a prediction problem, where the objective is to at any given time estimate the time between the last and the yet not arrived event. If $\Delta(k) = t(k) - t(k-1)$, using the assumed stationarity stochastic properties of n, a predictor can be written on the discrete time shift operator form

$$\hat{\Delta}(k+1) = \frac{B(q^{-1})}{A(q^{-1})}\Delta(k) \tag{5.10}$$

$$\hat{y}_i(k) = \frac{1}{\Delta(k+1)} \tag{5.11}$$

The selection of the polynomials B and A makes it possible to filter out specific noise components of the sequence and as long as the filter has unit stationary gain $(B(1)/A(1) = 1)$ the proper mean will be obtained. There is one caveat, however, when dealing with a decreasing rate. If the prediction states that an event should occur but there is none, the estimate must be updated to reflect this. This can be done through noting that if t time has passed since the last event occurred and $t > \hat{\Delta}(n+1)$, then the highest possible current rate would be sustained if an event would arrive at the time $t+\varepsilon$. A way to check for this is to tentatively update the prediction as if an event had occurred at the time t and check if the estimated rate would be lower. If $\Delta_e(k)$ denotes the extended sequence $\{...,\Delta(k-1),\Delta(k),(t-t(k))\}$, the resulting estimator for $y(t)$ would then be

$$\begin{aligned}
\hat{\Delta}(k+1) &= \frac{B(q^{-1})}{A(q^{-1})}\Delta(k) \\
\hat{\Delta}_e(k+1) &= \frac{B(q^{-1})}{A(q^{-1})}\Delta_e(k) \\
\hat{y}_i(t) &= \frac{1}{\max(\hat{\Delta}(k+1),\hat{\Delta}_e(k+1))}
\end{aligned} \tag{5.12}$$

Advantages with this approach is that the filter is fixed in time and space complexity. There is also the added degree of freedom in selecting the filter polynomials The downside is that badly chosen polynomials can yield a noisy estimate. By choosing $\deg(A) = 1$ and $\deg(B) > 1$, the filter gets a finite impulse response (FIR) structure.

As this approach has finite memory, just like the time window filter, it is also called an event window filter. If instead $\deg(B) = 1$ and $\deg(A) = 1$, the structure is called autoregressive (AR) or infinite impulse response filter (IIR).

k-parameter estimation

The execution rate model given by Equation (5.5) depends on the unknown parameter k, intended to be to be estimated in runtime. Given an estimate of the current execution rate $\hat{y}_i(t)$, falling back on equation (5.5) results in the following estimate:

$$\hat{k}_i(t) = \frac{\hat{y}_i(t)}{u_i(t-h_i)} \tag{5.13}$$

Unfortunately, this estimate does not take the disturbance v_i (from Figure 5.2) into account. As it is possible to measure $x_i(t)$ directly, a better estimator would be

$$\hat{k}_i(t) = \frac{\hat{y}_i(t)}{x_i(t_1) - x_i(t_0)}, \tag{5.14}$$

if t and t_0 and t_1 are such that the events used to form $\hat{y}_i(t)$ occur in the interval (t_0, t_1).

5.7 Extension into mixed domain models

One important objective of this work is to model a system where there is noticeable interaction between the software components and the hardware components. The approach taken is to see it as a system which performance is governed by the flow of resources. The concept of *resource* is here extended to mean a quantity, bounded and non-negative, that through a system component is converted into another resource. The system performance is then expressed in terms of this transformation.

A *resource flow* is the exchange of a resource between two components of the system. In a CPS, a flow can be physical (e.g., heat or power) or virtual (computations or data). In this work physical flows are considered to be $\mathscr{L}_2$ functions while virtual flows are either $\mathscr{L}_2$ or a sequence of Dirac-spikes, with finite density. Formally, let $\mathscr{U}$ denote the set of generalized functions on $\mathbb{R}$ such that if $u(t) \in \mathscr{U}$ and, $t_1, t_2 \in \mathbb{R}$ then

$$\int_{t_1}^{t_2} u(t)dt \tag{5.15}$$

exists and has the sign of $t_2 - t_1$. The rationale behind resource flows as Dirac-spikes is the event-like manner in which many important virtual resources are generated. As an example, let $n_{CPU}(t)$ denote the number of completed instructions in a CPU and $n_{task}(t)$ the number of times a sporadic task running on that CPU has executed. Obviously, $n_{task}(t)$ depends on $n_{CPU}(t)$. In order to analyze the performance of the task, e.g., to check if it completes cycle n before a certain time, it is necessary to

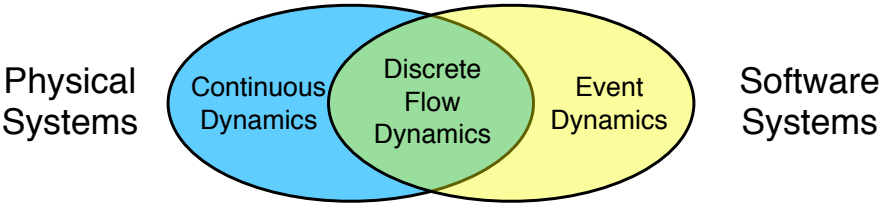


Figure 5.6 A diagram showing how the domains of physical and software systems have their own unique dynamics, but share the discrete flow dynamics.

study how $n_{task}(t)$ and $n_{CPU}(t)$ evolve over time, but due to their staircase nature, if t_0 denotes some specific time,

$$\lim_{h \to 0} \frac{n_{task}(t_0) - n_{task}(t_0 - h)}{h} \tag{5.16}$$

is either 0 or undefined. Therefore the dynamics of a system involving virtual flows must be expressed in discrete time, here called a *discrete flow dynamic*.

Forming such a model for physical systems can be done through sampling. Note however that while the sampled system description says nothing about the system behavior between sample points, signals like $n_{CPU}(t)$ and $n_{task}(t)$ are defined for all t. Because of this, it is possible to talk about exactly when the changes in n occur.

To summarize, the dynamics of a physical system is often modeled through differential equations. These can be discretized or sampled to give a discrete time model. Software system are usually modeled through event-based dynamics, but these can be re-formulated as discrete flow dynamics, comparable with the discretized continuous model. This gives a common model domain with which to express the entire system dynamics. Figure 5.6 shows how the two domains and the dynamic descriptions relate. How such a combined model can be used for resource management is explored further in Chapter 7.

6

Competitive resource management

Chapter 5 presented the performance of the system expressed in two primary metrics, execution rate and event timestamps, both dependent on allocated resources. This chapter introduces allocation through convex optimization as a way to maximize system performance in the execution-rate sense. A specialized allocation algorithm targeted at embedded real-time systems is presented together with a discussion about its convergence properties and real-time behavior. This chapter is based on the papers [Lindberg, 2009; Lindberg, 2010b; Lindberg, 2010a].

6.1 Allocation under resource constraints

The purpose of resource management is to maximize system performance, a non-trivial task when there is not enough resources to run all components at nominal execution rates. In the terminology introduced in Chapter 5, it is to be expected that we cannot keep the rate error of all components at zero, but have to compromise. To evaluate such a compromise, a global performance metric is needed. For the set of independent components, a natural choice would be an aggregate of the individual utility functions. The selection of such an aggregate is an important design parameter, which can be used to achieve a compromise suitable to the primary system use cases. For example, it can make sense to prioritize system services over 3rd party components or minimizing the worst case rate error.

In this thesis, it will be assumed that the system performance objective is on the form

$$J(u) = \sum_{i=1}^{N} J_i(u_i) \tag{6.1}$$

where J_i is the contribution from component i. In the presence of a resource constraint, $\sum u_i \leq U$, and under the assumption that resources are positive quantities,

the allocation problem can be posed as an optimization problem

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \sum_{i=1}^{N} J_i(u_i) \\ \text{subject to} \quad & \sum_{i=1}^{N} u_i \leq U \\ & u_i \geq 0, \forall i \end{aligned} \tag{6.2}$$

Convex allocation problems

In order to find efficient and reliable ways to solve the allocation problem online, i.e., without human supervision, (6.2) needs to be restricted. By limiting the component contributions J_i to differentiable convex functions, the resulting form will qualify as a convex optimization problem [Boyd and Vandenberghe, 2004]. This formulation has has several attractive properties:

- it will always be a feasible problem,
- it will have a unique solution,
- powerful theory exist for designing solvers, and
- it allows for general forms of system utility functions.

Recall the definition of the steady state rate error from Section 5.3:

$$e_i(u_i) = r_i - y_i = r_i - k_i u_i. \tag{6.3}$$

Since Equation (6.3) constitutes an affine mapping, any J_i taken as a the composition of a convex function and $e_i(u_i)$ will also be convex [Boyd and Vandenberghe, 2004, p. 79]. For example, $e_i(u_i)^4$ and $|e_i(u_i)|$ would be viable choices for J_i. Conceptually, this means that the problem becomes distributing the rate error rather than the resources.

When designing the objective function, some attention should be given to its sensitivity to parameter changes. Since parameters will be estimated online under noisy circumstances, adopting an linear programming-style objective can lead to sudden jumps in the solution. See [Maciejowski, 2002, p. 151] for a discussion on the merits of linear and quadratic objectives.

Disabling components

For some components performance is only tolerable when y_i is close to r_i. For this purpose a lower rate bound y_i' can be introduced, which represents the lowest relevant execution rate. It would then be desirable to allocate resources so that all components (if possible) are within their respective regions of tolerable performance. If

this is not possible, low priority components must be disabled until

$$\sum_{i=1}^{N} \frac{y_i'}{k_i} \leq U. \tag{6.4}$$

Solver design

Though parameters in this system might not change often, events such as the reconfiguration of an application, the arrival or deactivation of a component or a change in CPU resource availability in response to risk of overheating might require that we quickly redistribute resources. Therefore, an algorithm suitable to solving (6.2) online is needed. More specifically, it is desirable that it

1. takes minimal system resources,
2. accounts for changing parameters as quickly as possible,
3. produces results in deterministic time and memory,
4. can improve upon a previous allocation even if aborted before optimum was computed, and
5. is suitable for implementation in fixed point arithmetics.

6.2 Incremental optimization

This section proposes an algorithm which can solve (6.2) efficiently and with desirable time and memory characteristics. The central idea of the algorithm is to see the solution as a sequence of resource transfers between two components, in effect solving the problem as a series of one-dimensional problems. The benefit of this approach is that each step is computationally inexpensive, predictable in execution time and has numerical properties well suited for fixed point implementations.

The problem structure is very similar to the water filling approach to power distribution seen in the communications literature, see for example [Boyd and Vandenberghe, 2004], as illustrated in Figure 6.1. Essentially, the components are represented by a set of connected "tubes", with dimensions so that if the tube is filled with a quantity of water u_i, the resulting surface level will correspond to the value of the marginal utility function $P_i(u_i) = \partial J_i / \partial u_i$. For the example in Figure 6.1, $J_i(u_i) = (r_i - k_i u_i)^2, \forall i$. In this case the tubes will be $-2k_i r_i$ high and $1/2k_i^2$ wide.

Algorithms used to solve this type of problem are often based on the water filling principle, i.e., the solution is calculated as if water in an amount equal to the allocatable resource had been poured into the connected tube set and allowed to settle. The resulting water surface will then define the optimal allocation. The method presented in [Boyd and Vandenberghe, 2004] starts with empty tubes and then fills them until the resource is depleted. Mathematically this is done by solving

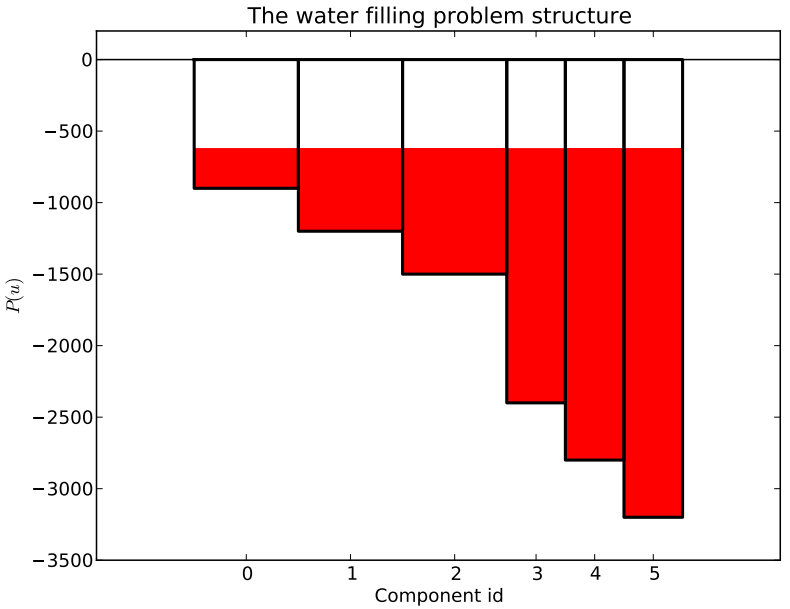


Figure 6.1 The structure of the water filling problem. The geometry of the "tubes" is defined by the cost function and the optimal solution is found as the water surface if the tubes are filled to a common level.

a series of linear equations. Two drawbacks of this scheme when applied to the embedded system resource problem is that

- the solver starts from scratch, thereby loosing information about the previous solution, and that
- the linear equation method only works for a specific case of utility functions.

The algorithm presented below also relies on the water filling principle, but works by equalizing water level between two components at a time. This accounts for heterogenous sets of utility functions and is also easier to implement on limited precision systems as the expressions to be evaluated are simpler.

Assume that two components $\mathscr{C}_i, \mathscr{C}_j$ are picked from the set during the k:th step of the algorithm. Let J^k be the cost at the beginning of the step and $J^k_{i,j}$ denote the contribution by $\mathscr{C}_i, \mathscr{C}_j$ to J^k. Consider now what happens if an amount of resource δ is transferred from $\mathscr{C}_i$ to $\mathscr{C}_j$ so that their combined contribution to J^{k+1} is minimized,

i.e., by solving

$$\begin{aligned} \underset{\delta}{\text{minimize}}\ J_{i,j}^{k+1} = \underset{\delta}{\text{minimize}} \quad & J_i(u_i^k + \delta) + J_j(u_j^k - \delta) \\ \text{subject to} \quad & -u_i^k \leq \delta \leq u_j^k \end{aligned} \tag{6.5}$$

This ensures that

$$J^{k+1} \leq J^k \tag{6.6}$$

In other words, by in each step solving a subproblem, performance will improve incrementally. Solving this minimization subproblem for general convex functions $J_i(u_i)$ can be done by modifications to unconstrained methods such as Newton-Rhapson or even bisection.

One notable choice of $J_i(u_i)$ is $w_i e_i(t_u)^2$ in which case the allocation problem becomes a special case of quadratic programing (QP). The quadratic form is attractive for several reasons, including that the subproblem can be solved in two simple steps and being less sensitive to small parameter changes than, e.g., linear programming (LP). Let

$$\begin{aligned} J_i(u_i) &= w_i(r_i - k_i u_i)^2 \\ J_j(u_i) &= w_j(r_j - k_j u_j)^2 \\ \delta = \underset{\delta}{\arg\min} \quad & J_i(u_i + \delta) + J_j(u_j - \delta) \\ \text{subject to} \quad & -u_i \leq \delta \leq u_j \end{aligned} \tag{6.7}$$

As $J_{i,j}$ is a convex function, if it has an unconstrained minimum that violates the constraints on δ, the constrained minimum is found by picking δ on the constraint. The solution to (6.7) is then calculated as

$$\begin{aligned} \delta_{nc} &= \frac{w_i k_i r_i - w_i k_i^2 u_i + w_j k_j^2 u_j - w_j k_j r_j}{w_i k_i^2 + w_j k_j^2} \\ \delta &= sat(\delta_{nc}, -u_i, u_j) \end{aligned} \tag{6.8}$$

Selecting the pair $\mathscr{C}_i, \mathscr{C}_j$ for each step is the other part of the algorithm. The proposed strategy is derived from the Karush-Kuhn-Tucker (KKT) conditions (see for example [Boyd and Vandenberghe, 2004]). Posing (6.2) on standard form, the Lagrangian becomes

$$L(u, \lambda, \nu) = \sum_{i=1}^{N} J_i(u_i) + \sum_{i=1}^{N} -\lambda_i u_i - \nu(U - \sum_{i=1}^{N} u_i) \tag{6.9}$$

The KKT-conditions state that $\nabla L(u, \lambda, \nu) = 0$ in an optimal point. By studying the expression

$$\frac{\partial L(u_i, \lambda, \nu)}{\partial u_i} = \frac{\partial J_i(u_i)}{\partial u_i} - \lambda_i + \nu = 0 \tag{6.10}$$

it can be seen that in an optimal point, either $u_i = 0$ or $-\partial J_i(u_i)/\partial u_i = \nu$. Recall that $P_i(u_i) = \partial J_i(u_i)/\partial u_i$. If $u_i = 0$ and therefore $\lambda_i > 0$, then $P_i(u_i)$ must be less than ν. In other words, a point where $P_i(u_i) > P_j(u_j)$ and $u_j > 0$ does not minimize (6.5).

- If the algorithm tries to select $\mathscr{C}_i, \mathscr{C}_j$ so that $P_i(u_i) > P_j(u_j)$ and $u_j > 0$, solving (6.5) results in $J^{k+1} < J^k$.

- If there is no such pair to select, then that point satisfies the KKT-conditions of (6.2) and the allocation is optimal.

It follows that such a strategy will make the algorithm converge to the optimum. The convergence speed will obviously depend on the specific transfer sequence. As the intended domain is real-time allocations, an efficient strategy is needed. It is desirable that each step reduces $J(k)$ as much as possible and from (6.5) it is evident that the size of the gain depends on

- the difference in $P(u)$ between the two components and

- the amount of resource available to redistribute.

The two criteria can be in conflict, particularly if there are large variations in k_i. However, it is here assumed that if a component requires much less resources then the others, it does not have to be part of the optimization. Rather, such a component will be seen as part of the background noise, denoted v_i in Section 5.3.

An intuitive strategy for picking the transfer pair is to sort the components according to $P_i(u_i)$ and select the two furthest apart, skipping those with highest $P_i(u_i)$ for which $u_i = 0$. The intuition behind this can be seen if $P_i(u_i)$ is interpreted as the water level, as the sought common surface must lie between the extremes. The proposed implementation uses a red-black tree that makes finding the pair an $O(1)$ operation and inserting them back after the transfer an $O(\log n)$ operation (see for example [Cormen, 2001] for complexity analysis of red-black trees). As the algorithm uses an iterative loop and the persistent data allocated scales linearly with the problem, memory need for a system with a known size can easily be calculated.

To illustrate the workings of the algorithm, consider a case with three components $\mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3$, and $J_i = e_i(u_i)^2, i = 1,2,3$. The components have the properties

i	r_i	k_i	u_i
1	55	50	1
2	25	60	0
3	20	60	0

Figures 6.2 and 6.3 illustrates how the algorithm then operates during the first iterations. Though this example uses the same form of cost function for all components, it is worth noting that the algorithm allows for any mix of convex cost functions.

This could be useful for a system designer when distinguishing between how resources are allocated to, e.g., system services and 3rd party add-ons.

In the initial state, $\mathscr{C}_1$ is best off, seen by the high P_1 while P_2 is worst off. The first transfer then equalizes the potentials P_1 and P_2. As can be seen in Figure 6.3, the system cost decreases rapidly in the beginning. The performance resulting from a fair allocation, as defined by [Demers et al., 1989], is provided for comparison. The fair allocation will in general terms allocate an equal amount of resource to all components, which will be unfavorable for components with relatively low k and high r, such as C_1.

Figure 6.4 exemplifies a scenario where the problem parameters change over time, assuming allocation is recomputed every second. At 33 seconds, r_2 changes to 40, exemplifying something that could be caused, e.g., by an internal mode switch or a user command. The algorithm will here reduce the performance of $\mathscr{C}_1$ and $\mathscr{C}_3$ slightly. At 66 seconds, the total resource level drops by 25%, which could happen for instance if the system was overheating and the CPU needed to be throttled to reduce heat generation. In this case, $\mathscr{C}_3$ is shut down, which may or may not be the intention of the system designer. Constrained allocation in this manner is not starvation free and care must be taken when choosing the component cost functions.

6.3 Experimental results

A series of experiments were run, using the algorithm to find an allocations for random component sets under overload conditions (i.e., $\sum_{i=1}^{N} r_i/k_i \geq U$). The aim with the simulations were to show the computational efficiency and get a feel for the convergence rate. The simulations were run on an 2.40 Ghz Intel Pentium(R) 4 based computer with 512Mb memory which was running Linux 2.6.27. The compiler used was gcc 4.3.2 using the -O3 compiler flag. The experiments use the $J = ||e(u)||_2$ cost function.

Figure 6.5 shows the iteration time as function of the number of components and in Figure 6.6 we see the termination time of the optimization. Each iteration consists of picking the two components, calculating the transfer and resorting and the algorithm terminates when the solution is within a set tolerance of the optimum. The variance comes primarily from sorting artifacts and cache dynamics. Component sets were generated randomly and ran 10 times in succession. As a comparison number, a two variable QP problem with the structure of (6.2) took 500 ms to solve with a general QP solver written in C++ [*CGAL* 2010] on the same computer as used in the other experiments. This is most likely due to the larger overhead for initializing the algorithm, something that will make it resource expensive to use for a problem where parameters and problem structure change over time.

Studying Figure 6.5, iteration time appears to increase linearly with the problem size. This seems counterintuitive as the red-black tree is performs with $log(n)$-like complexity. However, as the number of components grow large, it becomes increas-

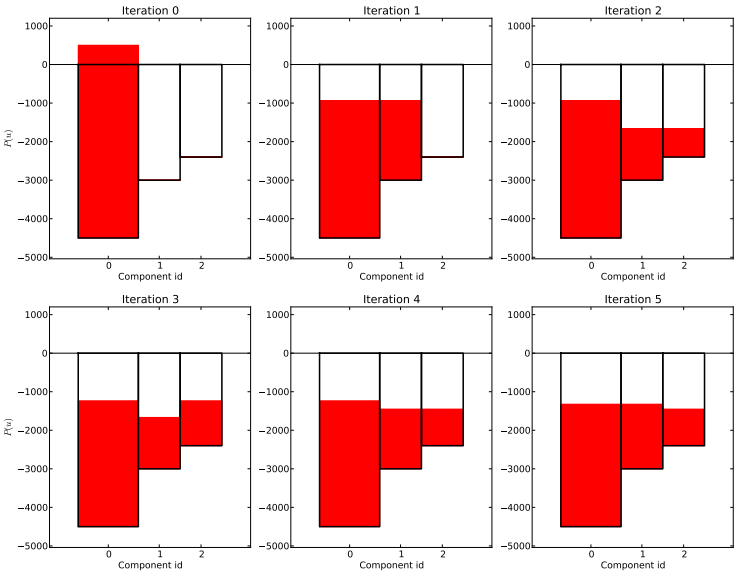


Figure 6.2 A sequence of allocation iterations, showing how the marginal utility level is equalized by pairwise transfers.

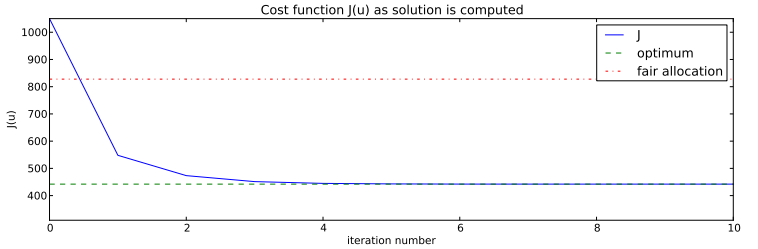


Figure 6.3 A plot of how the cost function decreases in value for each iteration in the case illustrated in Figure 6.2, with the performance resulting from a fair allocation provided for comparison.

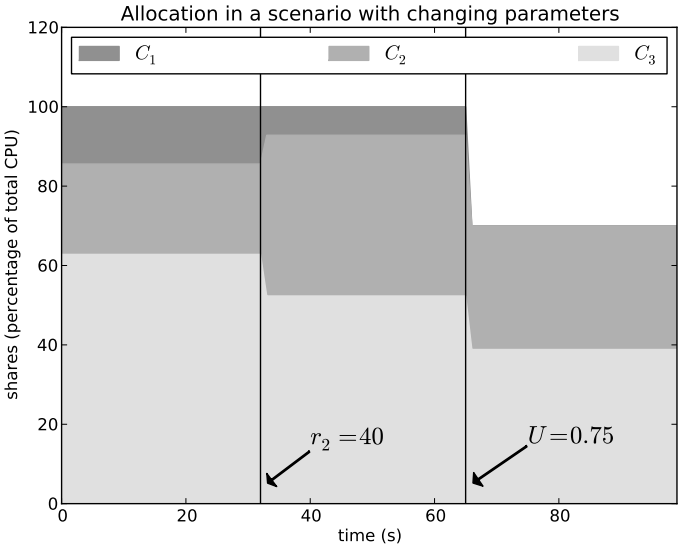


Figure 6.4 A scenario where the problem parameters change over time, assuming allocation is computed every second. At 33 seconds, r_2 changes to 40 and at 66 seconds, the total available resource level drops by 25%.

ingly likely that some of them will be allocated zero resources. Finding a component with high $P_i(u_i)$ will then involve a linear search, which would explain the trend in iteration time. As the component set grows larger, the variance in iteration time grows. This is because the sorting operations when re-inserting components into the red-black tree become more and more expensive. It is also to be expected that keeping the data structures in cache memory will be increasingly difficult for large problems.

The trend in termination time is more according to intuition. As iteration time grows linearly and the number of iterations must grow at least as fast as problem size, the algorithm is at best quadratic in complexity.

With a solver that can determine an optimal allocation in milliseconds, periodic use of optimization in embedded system resource management is feasible.

6.4 Implementation

In order to assess the feasibility of using the above described allocation method in a more realistic case and to provide insight into platform design problems, a prototype implementation was created using the Linux CFS scheduler as RBS mechanism.

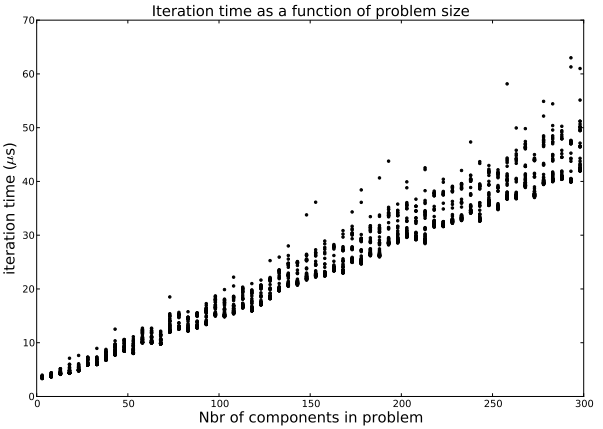


Figure 6.5 Measurements of iteration times. Optimization for each component set was run 10 times. The variance comes from a combination of sorting artifacts and cache dynamics.

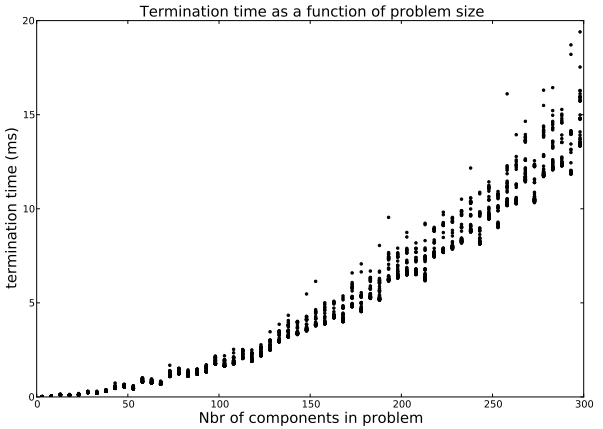


Figure 6.6 Measurements of optimization termination time. Optimization for each component set was run 10 times. Variance is due to sorting artifacts and cache dynamics.

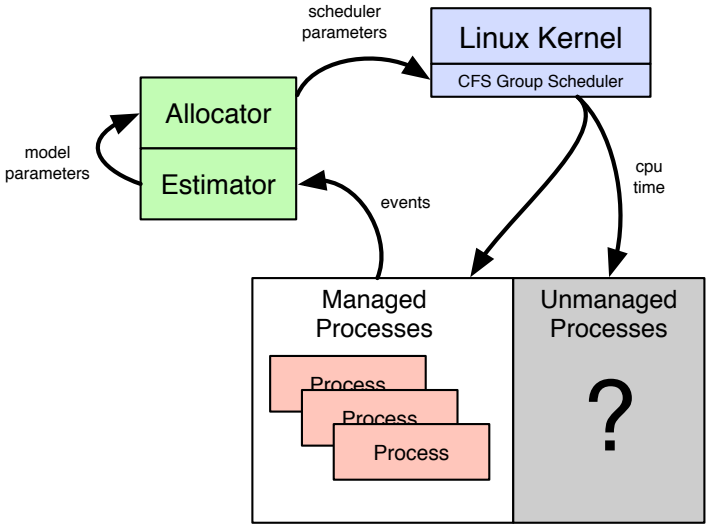


Figure 6.7 Resource management architecture.

The setup included software components in the form of processes with synthetic workload and a resource manager implemented as a standalone application in C. The framework was developed with special care not to make use of specialized kernel patches or extensive external libraries, as such dependencies could result in portability issues. It also serves as a demonstration that resource management can be employed using an off the shelf OS, which is especially important for consumer products where time-to-market is an important consideration.

6.5 Resource management architecture

Figure 6.7 shows a schematic of the system. The blue block signifies a standard Linux kernel with CFS group scheduling capabilities (v2.6.24 or later up to at least v2.6.35).

The green block represents the resource management framework, which in turn consists of an allocator and estimator. These two are contained within the same process for data sharing purposes but as two separate POSIX threads.

The red blocks represent processes that are managed by the framework, meaning that they each execute in a reservation and supply the estimator with data. It is assumed that these account for the majority of the resource consumption in the system.

MIPC - a minimalistic IPC protocol

Data is transmitted between components using a Minimalistic InterProcess Communication (MIPC) protocol developed for use with the resource manager. Data is sent as datagrams and MIPC supports both local UNIX sockets and IP sockets. As the estimation techniques support missing measurements, the potential overhead of using TCP-based connections can be eliminated.

MIPC itself only handles sending raw byte data so the client will need to specify interfaces on top of that, typically by sharing C-struct definitions. The important MIPC-operations are

- `mipc_connect_server`
- `mipc_connect_client`
- `mipc_send`

and the full API is found in Appendix A. The protocol supports the bare minimum required for the framework and for comparison, the compiled binary is less than 10 kbytes in size and only have dependencies to the C standard library and libc, while, e.g., D-Bus, a popular interprocess communications mechanism in desktop Linux distributions [*D-Bus* 2010], is over 100 kbytes in size and has dependencies to over 1 Mbytes of additional libraries.

Unmanaged processes

It is here assumed that the majority of the resource requirements comes from a subset of the running processes. The remaining is seen as noise. Should these components require a noticeable amount of resources, this will affect the estimate of the relation between the allocated share and the resulting execution rate, thus effectively make the CPU seem slower.

Processes or threads

Keeping a component based abstraction level is an important goal, as this gives increased flexibility to the software designers. By using processes to track resource usage rather than threads, the anatomy of the components is hidden from the resource manager. Whether a process consists of one or several threads of execution should not matter in this framework.

Experiment setup tools

A range of Python based tools were developed in order to facilitate the scripting of test scenarios. A Python implementation of MIPC is used to signal processes. For operations that require root level privileges and access to non-standard system calls, a few C-based utilities were created and then wrapped in Python code.

6.6 Measuring time and resource consumption

The framework requires the managed components to push data to the estimation algorithm in the form of cycle completion events. These events contain

- a time stamp and
- a reading of the accumulated CPU-time for the component.

The resource measurement is performed as the time stamp is taken, as sending and processing introduce latencies. It also means that the resource cost of measuring is factored into the overall component resource requirements, thereby distributing the overhead rather than centralizing it on the estimator.

Measurements of resource consumption have been done using the system call `clock_gettime()` rather than `getrusage()` as it provides better precision. An alternative would to be use the cpuacct-subsystem available with control groups, but the system call was chosen as it is both simpler to use and more efficient than parsing text files.

Reservations with CFS Group Scheduling

The group scheduling functionality (see Section 3.2) that was introduced with Linux v2.6.24 offers an easy to use way to do soft reservations. From an experimental point of view, this requires some extra effort as the allocation theory developed assumes hard reservations.

Shares are calculated by the algorithm as percentage of the total CPU capacity. This must then be translated into an integer number, as used by CFS, in such a way that the desired share equals the integer weight / the sum of all weights. To avoid quantization problems, the total available shares have been selected to be 10000. The CFS scheduler does not allow for an individual reservation period to be set on a process basis so apart from share, no additional parameters must be set.

Effectuating the allocation requires writing the shares to the virtual files in a control group type file system.

Synthetic components

In order to run experiments with relevant load profiles, a synthetic component was implemented and instrumented with logging and a MIPC-based reconfiguration interface. The components execute in accordance with the cyclic component model (see Section 5.3) and with parametrized behavior. Cycle time is drawn randomly from an interval specified at startup and changed periodically. The parameters that govern the behavior are

- **k_min, k_max** - controls the distribution of the cycle times.
- **change_interval** - determines how often cycle times are randomized.

- **rate** - the rate set-point.

The components also log all completed cycles together with a snapshot of the parameter set.

Estimator implementation

The estimation is a passive component in this framework and updates estimates only when needed, in this case triggered by the allocation thread. A MIPC-server is set up to collect all incoming data events from the managed components, which is then stored in a record for each individual client. The client submits a unique identifier, in this implementation the process id of the main component thread, which is then used as a primary key in a table containing all the managed components.

The implementation supports individual estimator functions for each component and supplies three default methods for rate estimation:

- time window (counting events over an interval)
- event window (FIR-structure, time for a fixed number of events)
- autoregressive filter of the event arrival intervals (IIR-structure)

As all events are stored, the estimation history can be replayed at the conclusion of the experiments and compared with the component log files.

Allocator implementation

The allocator is implemented as a periodic thread running at 1 Hz. The work order is

- pull parameters from the estimator
- iterate the incremental solver from Section 6.2 until the potentials are within the tolerance level
- effectuate the allocation by writing to the cgroup file system

The allocation algorithm utilizes a red-black b-tree [Cormen, 2001] to sort the components by potential, which allows for robust performance. It is straightforward to both add and remove tasks from the tree, thereby allowing the structure to persist between iterations. This circumvents the often heavy set-up portions of off-the-shelf solvers.

A system parameter is the termination threshold. The algorithm checks the difference in potential between the highest and lowest level and if they are close enough, the solver terminates. In the simulations used for this thesis, the tolerance is set to 0.001.

6.7 Example runs

This section presents the results from a sequence of experiments run on a desktop Linux computer. The hardware was a 2.4 Ghz Pentium 4 with 512 Mb of main memory running a Linux 2.6.27-based Debian system. The background noise consists of the software that runs on a typical Debian desktop, including the X11/Gnome graphical environment, as well as an Apache web server and a MySQL database engine.

Estimation and control

The first experiment is to validate the estimation and control strategy. A single software component with constant but unknown cycle execution time is here controlled using feedforward based on an estimation of the k-parameter.

Figures 6.8 and 6.9 show the scenario using one second time window estimation and a 15 long event window estimation respectively. The time window performs well at higher rates, having more information to form the estimate, while the event window estimator is more responsive to change. The AR-approach proved to be difficult to tune, resulting in much noisier estimates, and was not used for the experiments.

Constrained allocation

In this scenario, three components are running in a situation where the system is overloaded. The components change their cycle execution rates randomly every 3 seconds, with k_max/k_min equal to 2. Figure 6.10 shows the estimated rates over time and Figure 6.11 displays the cost for the dynamic convex allocation (DCA) with a comparison with static worst case allocation (SWA), i.e., the performance attained by only allocating resources based on the worst case execution time. Towards the end of the experiment, the component set is close to their worst case cycle times, and this causes the system cost to approach that of the worst case allocation.

To study how the performance depends on the level of uncertainty, a series of experiments were run with increasing k_max/k_min. In scenarios with low uncertainty, the dynamic strategy performs worse than the static alternative, as the estimated parameters will, due to the presence of noise, never be completely correct. However, as uncertainty grows, the dynamic strategy shows significant performance advantages, which is shown in Figure 6.12.

6.8 Conclusions

This chapter has introduced a method for dynamic resource allocation based on convex programming and a rate based utility model for cyclic software components. An algorithm designed to efficiently solve the resulting optimization problem on

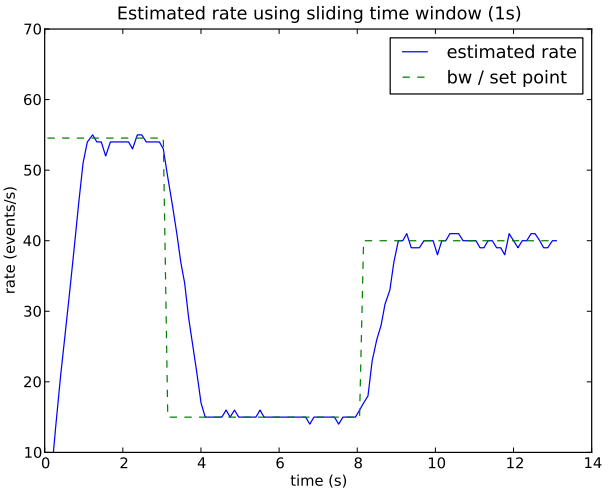


Figure 6.8 Estimation and control using time window

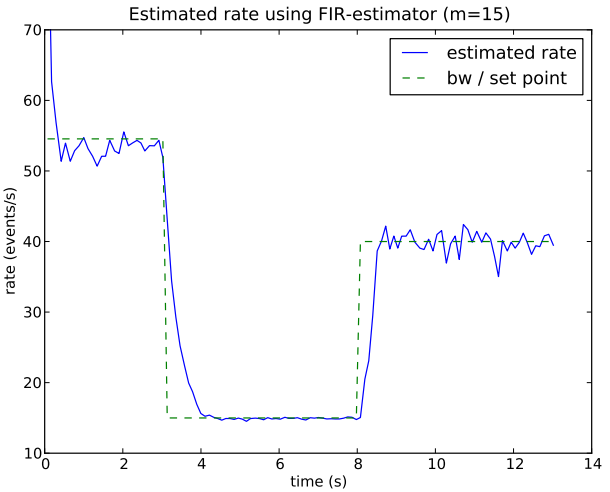


Figure 6.9 Estimation and control using event window

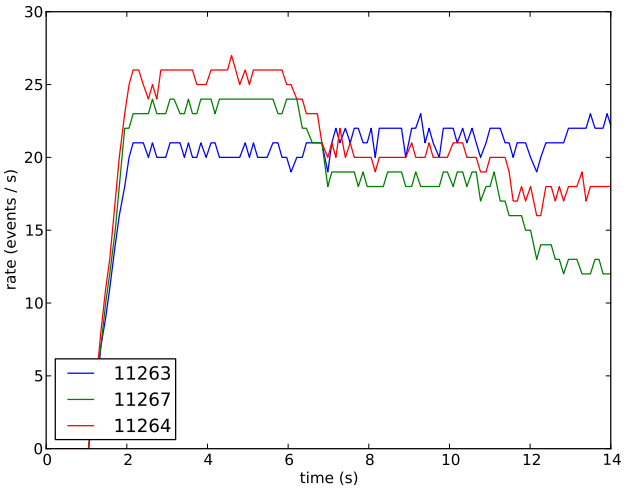


Figure 6.10 Allocation in an over-utilized system with rates estimated using a one second time window.

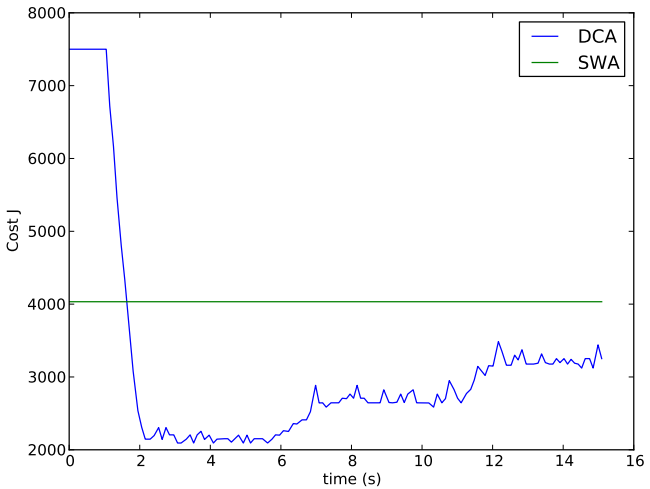


Figure 6.11 Cost for Dynamic Convex Allocation (DCA) vs Static Worst case Allocation (SWA) over time.

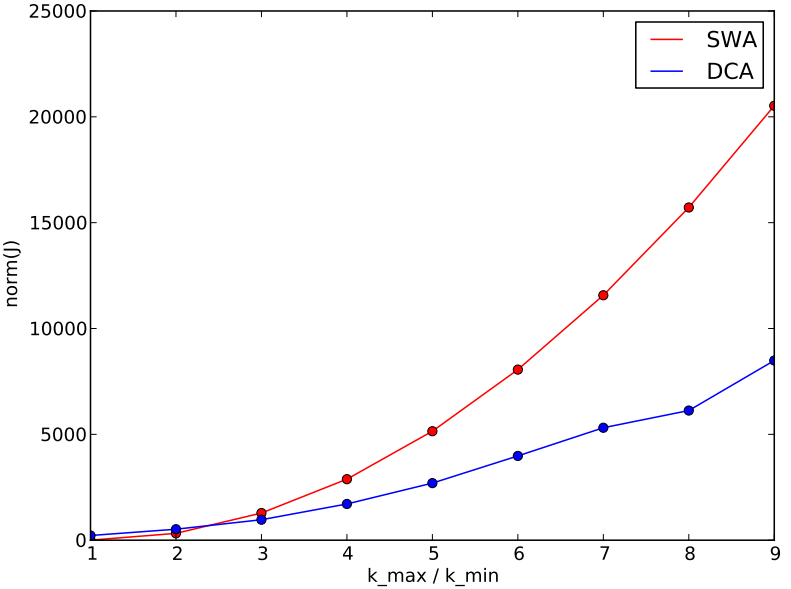


Figure 6.12 Average Cost for Dynamic Convex Allocation (DCA) vs Static Worst case Allocation (SWA) as uncertainty grows.

limited precision hardware has been presented together with analysis of its runtime performance.

Results from experiments run on an implementation of the resource management scheme on Linux has are presented with comparisons to the corresponding worst case-based performance.

7

Collaborative resource management

This chapter discusses the application of feedback control to a system where component are dependent on each other. More specifically, the results presented here pertains to systems where the performance metrics are functions of how the components collaborate around tasks, hence the term *collaborative resource management*. As a result, the theory presented does not build on Chapter 6, and instead presents an alternative scenario where adding resources to a component may even reduce system performance. The component model used is, however, still based on Chapter 5. The thermal control approach is further validated by an implementation on a mobile robot. The chapter is based on [Lindberg and Årzén, 2010b; Romero Segovia et al., 2011].

7.1 Allocation vs feedback

One of the key simplifications made in the resource allocation algorithm proposed in Chapter 6 was to optimize the performance in a stationary sense, as introducing state dynamics into the optimization would require a more comprehensive solver. As such, the allocation strategy used in this thesis is largely a feedforward solution and therefore blind to performance metrics explicitly connected to state information.

Given perfect information about execution times and resource availability, these metrics could be controlled through the allocation strategy. However, as a central theme of this work is the presence of uncertainty and disturbances, it must be assumed that this is not possible. Specifically, it can be expected that transient phenomena will occur due to disturbances and structural changes in the system, such as hardware interrupts and the activation or reconfiguration of components. Assuming that these occurrences cannot be predicted, they must be addressed through feedback.

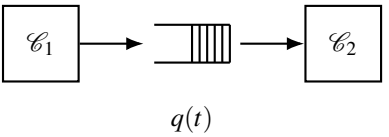


Figure 7.1 Two components connected through a FIFO-queue. $q(t)$ signifies the number of elements in the queue.

7.2 State related performance metrics

The two aspects of performance that will be discussed here are

- integrator dynamics and
- state transition events.

Queues and integrators

Asynchronous communication between components in software systems is commonly done through FIFO-queues. This is practiced both in consumer grade multimedia frameworks such as GStreamer [Taymans et al., 2010] and more academically oriented actor-based languages such as CAL [Bhattacharyya et al., 2009; Eker and Janneck, 2003]. A common problem that arises in such designs is the regulation of queue sizes, as this affects both end-to-end computational latency and storage requirements.

If a queue constitutes the connection from component $\mathscr{C}_1$ to $\mathscr{C}_2$, as illustrated in Figure 7.1, and $q(t)$ denotes the number of queue entries at time t, its resource dependent dynamics could be described by

$$\begin{aligned} y_1(t) &= k_1 u_1(t) \\ y_2(t) &= k_2 u_2(t) \\ q(t+h) &= q(t) + h(y_1(t) - y_2(t)) \end{aligned} \tag{7.1}$$

i.e., a discrete time integrator. From an allocation point of view, an imbalance between y_1 and y_2 results in either starvation of $\mathscr{C}_2$ or unbounded queue length. Therefore, the feedforward strategy should strive for $y_1 = y_2$. Transient effects can then be attenuated through feedback.

State transitions

State models are commonly used for software systems, defining the behavior in terms of states and state transitions. A state transition is caused by some internal or external dynamic and the passing from one state to another represents an event that is significant to the software. This could for instance be the completion of a computation, e.g., the decoding of a video frame or the termination of an optimization.

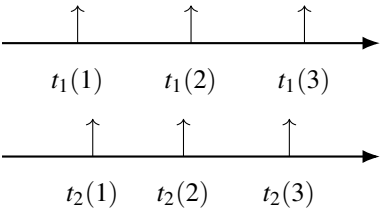


Figure 7.2 Synchronization between two event sequences is one possible objective that can be solved using feedback resource management. The synchronization error, $e_s(k) = t_1(k) - t_2(k)$, is difficult to address using the feedforward strategy as this relies on collecting information over a number of events.

This thesis specifically considers changes in the number of completed component cycles n_i, which are signified by the cycle completion events.

Given a number of state transitions over the time period h, as controlled through the allocation strategy, some applications will be sensitive to exactly when these occur. Keeping them periodic, i.e., equidistant in time, is a common goal, essentially the objective of classic scheduling techniques applied to, e.g., controller implementations. Minimizing jitter, guaranteeing the periodicity of sensor measurements and control actions fall under this class of problems.

Another possible scenario is to synchronize the state transitions between two components. This situation could arise when synchronizing sensors readings, such as audio and video capture, or as a strategy to reduce the complexity of a larger control problem. The synchronization problem could be seen as a generalization of the deadline problem, as the latter would arise if one sequence is set deterministically.

Let $t_1(k)$ and $t_2(k)$ denote two event sequences, as illustrated in Figure 7.2, generated as cycle completion events by the components desired to be to synchronized. Equation 5.6 describes the cycle dynamics

$$t(k+1) = t(k) + \frac{C(k)}{u(t(k))}, \tag{7.2}$$

the synchronization error $e_s(k)$ is defined as

$$e_s(k) = t_1(k) - t_2(k) \tag{7.3}$$

As with $q(t)$ in Equation (7.1), perfect allocation would result in no synchronization error, but given transient disturbances, a feedback approach could be employed to drive it to 0.

7.3 Hardware resources

Availability of the CPU resource is limited by hardware performance. Power and heat are two types of constraints in this setting. In this thesis the problem of thermal management is considered.

Normally a CPU can only operate properly if the temperature is kept below a certain level but if there is no active cooling, as is the case in many embedded platforms, this must be respected through control of CPU power. The options to do this include voltage- and frequency scaling and idling (e.g., executing the HLT instruction [Intel, 2010]), the last of which will be used here. The main reason for this is that the speed of the CPU directly affects the component model parameters and any on-line estimates would then change due to control action. Controlling the temperature through the resource level U and then imposing the limit

$$\sum_{i=1}^{N} u_i \leq U \tag{7.4}$$

simplifies the estimation strategy. Control can be effectuated through limiting the available CPU-time using an RBS framework. The resource level U is then determined by the thermal control algorithm, which becomes a part of the resource controller.

Thermal control

Given that the temperature is modeled with first order dynamics with slow disturbances, a PI-controller [Åström and Murray, 2008] is a simple and effective choice, though anti-windup measures must be added to handle effects from control signal saturation.

The pure PI-controller is defined as

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau) \tag{7.5}$$

By discretization of the dynamics using a forward Euler approximation, constraining the control signal U to the interval $[0,1]$ and adding an anti-windup tracking term to the integral part, the resulting control algorithm, described as pseudo-code, is

```
e := r - y_T;
V := K*e + I;
U := sat(V, 0, 1);
if (Ti > 0) then
    I := I + K*h*e/Ti + K*h/Tr*(U - V);
else
    I := 0;
endif
```

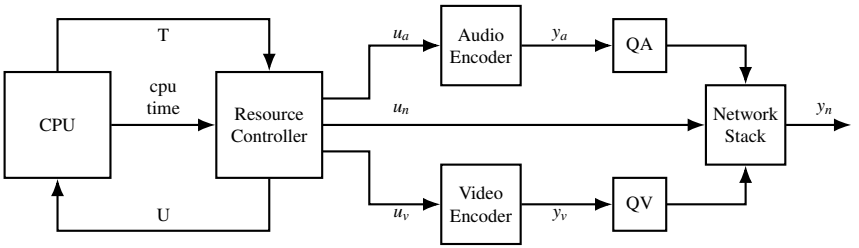


Figure 7.3 An overview of the conversational video pipeline

where h is the sampling interval, K, T_i are the PI-controller parameters and T_r is the anti-windup tracking gain.

7.4 Case study — Encoding Pipeline

The prototypical system to be considered is a conversational video pipeline as displayed in Figure 7.3. The software part consists of three tasks, an audio encoder, a video encoder and a network stack. The encoders are assumed to have private access to capturing hardware and it is also assumed that they are capable of variable rate execution. The encoders are connected to the network through the FIFO-queues QA and QV. In order to send a network packet, the stack requires one frame of audio and video.

In order to evaluate the performance of this system, the following metrics are defined

- **Sync error.** It is disturbing to the human eye when video and audio are out of sync and therefore it is natural to consider the difference in encoding timestamp between the corresponding audio and video frames. If both tasks are assumed to be cyclic and $t_a(k)$ and $t_v(k)$ denote the encoding timestamps for audio and video frames respectively, then

$$\begin{aligned} t_a(k+1) &= t_a(k) + \frac{C_a(k)}{u_a(t(k))} \\ t_v(k+1) &= t_v(k) + \frac{C_v(k)}{u_v(t(k))} \end{aligned} \tag{7.6}$$

  would be the corresponding dynamics. The sync error $e_s(k)$ is then defined as

$$e_s(k) = t_a(k) - t_v(k) \tag{7.7}$$

- **Latency.** Delay in the conversation is also an important quality metric and for this set up this will be the end to end latency, i.e., the delay from capture

to network. If $q_a(t)$ and $q_v(t)$ are the number of elements at time t in QA and QV respectively, then let the average encoding latency $e_l(t)$ be defined as the sum of the encoding delay $D_{a,v}$ and the network delay D_n.

The encoding delay is modeled as

$$D_{a,v} = \frac{\frac{C_a(k)}{u_a(t_a(k))} + \frac{C_v(k)}{u_v(t_v(k))}}{2} \tag{7.8}$$

which is the the average of the audio encoding time and video decoding time and the network delay as

$$D_n = (\frac{q_a(t) + q_v(t)}{2} + 1)\frac{C_n(k)}{u_n(t_n(k))} \tag{7.9}$$

i.e., the time it takes to process the queue backlog plus one cycle time for the packet itself. $e_l(t)$ is then defined as

$$e_l(t) = D_{a,v} + D_n \tag{7.10}$$

or in words, the computational delay combined with the network backlog in the queues.

- **Queue dynamics.** Using the cycle dynamics expressed in Equation (7.1), the dynamics of q_a and q_v is modeled as

$$\begin{aligned} q_a(t+h) &= q_a(t) + h(y_a - y_n) \\ q_v(t+h) &= q_v(t) + h(y_v - y_n) \end{aligned} \tag{7.11}$$

Control design

Latency It follows from (7.10) and (5.7) that latency can be controlled through minimizing the queue lengths and then keeping a uniform steady state cycle time across all components. The approach taken in this work is to combine feedforward control based on the total amount of resource with feedback *re-allocation* to reduce queue length.

The nominal feedforward controls are computed by combining the constraint that $u_a + u_v + u_n \leq U$ from Equation (7.4) with

$$y_a = y_v = y_n = y, \tag{7.12}$$

i.e., a steady state where the execution rates y_a, y_v and y_n are the same. From this it follows that

$$\frac{1}{k_a} + \frac{1}{k_v} + \frac{1}{k_n} = \frac{U}{y} \tag{7.13}$$

giving that

$$y = \frac{k_a k_v k_n}{k_a k_v + k_a k_n + k_v k_n} U, \tag{7.14}$$

which results in the control laws

$$\begin{bmatrix} u_a^{\text{ff}} \\ u_v^{\text{ff}} \\ u_n^{\text{ff}} \end{bmatrix} = \begin{bmatrix} k_a^{-1} \\ k_v^{-1} \\ k_n^{-1} \end{bmatrix} \frac{k_a k_v k_n}{k_a k_v + k_a k_n + k_v k_n} U. \tag{7.15}$$

To control the queue lengths, the feedforward controls are then modified with a feedback term u_q. As the control system cannot violate (7.4), u_q will be applied as

$$\begin{bmatrix} u_a \\ u_v \\ u_n \end{bmatrix} = \begin{bmatrix} u_a^{\text{ff}} \\ u_v^{\text{ff}} \\ u_n^{\text{ff}} \end{bmatrix} + \begin{bmatrix} \frac{k_v}{k_a + k_v} \\ \frac{k_a}{k_a + k_v} \\ -1 \end{bmatrix} u_q \tag{7.16}$$

subject to the constraints that the resulting controls $u_i \geq 0$.

To calculate $u_q(t)$, the closed loop dynamics of the queues are evaluated. It is assumed that the queues will be of equal length in steady state (see the section on sync error) so the feedback can be designed with any one in mind. Recall that

$$q_a(t+h) = q_a(t) + h(y_a - y_n) \tag{7.17}$$

Assume that the objective is to drive the queue length to some predetermined length r (for example zero). To achieve proportional control, let

$$h(y_a - y_n) = K_q(r - q_a(t)) \tag{7.18}$$

This converges to r for all $0 < K_q < 2$. Let u_q denote a feedback term, by which some of the resources allocated to the network stack is transferred to the audio- and video decoders, thereby regulating the relative rates of queue item production and consumption. Substitute

$$y_a = k_a(u_a^{\text{ff}} + u_q \frac{k_v}{k_a + k_v})$$
$$y_n = k_n(u_n^{\text{ff}} - u_q)$$

and solve for u_q to obtain the actual controls.

Sync error If C_a/u_a is approximated by $(k_a u_a)^{-1}$, it follows from the definition (7.7) that

$$e_s(k+1) = e_s(k) + (k_a u_a)^{-1} - (k_v u_v)^{-1} \tag{7.19}$$

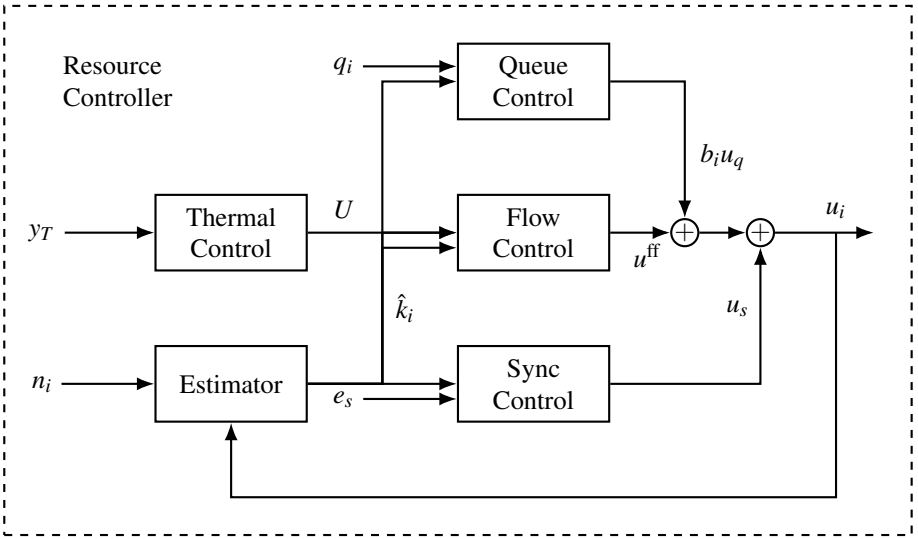


Figure 7.4 The resulting control structure including estimation, feedforward flow control based on the estimated model parameters and the available CPU resource and feedback based on queue state and sync error. b_i refers to the coefficients in Equation (7.16).

As there is a finite combined flow of CPU resource to the audio and video encoder, the approach taken here is to introduce u_s as a feedback term, modifying the feedforward allocation so that

$$e_s(k+1) = e_s(k) + (k_a(u_a+u_s))^{-1} - (k_v(u_v-u_s))^{-1} \tag{7.20}$$

is driven towards zero. While this seems to interfere with the queue control, the difference in time scale between the event-to-event dynamics makes its effects on the slower queue controller negligible. In fact, it is seen in Section 7.5 that controlling the sync error actually greatly simplifies the queue control. For proportional control, let

$$(k_a(u_a+u_s))^{-1} - (k_v(u_v-u_s))^{-1} = K_s e_s(k) \tag{7.21}$$

Under deterministic circumstances the sequence $e_s(k)$ will converge to zero for any $K_s \in (0,-2)$. Given the variations in the cycle execution times, some care should be taken when selecting K_s so that noise is not unnecessarily amplified. As this work is done without a detailed noise model, K_s is chosen conservatively as -0.5. The resulting equation is then solved under the constraint that $u_a+u_s \geq 0$ and $u_v-u_s \geq 0$ to obtain u_s.

The resulting control structure is presented in Figure 7.4.

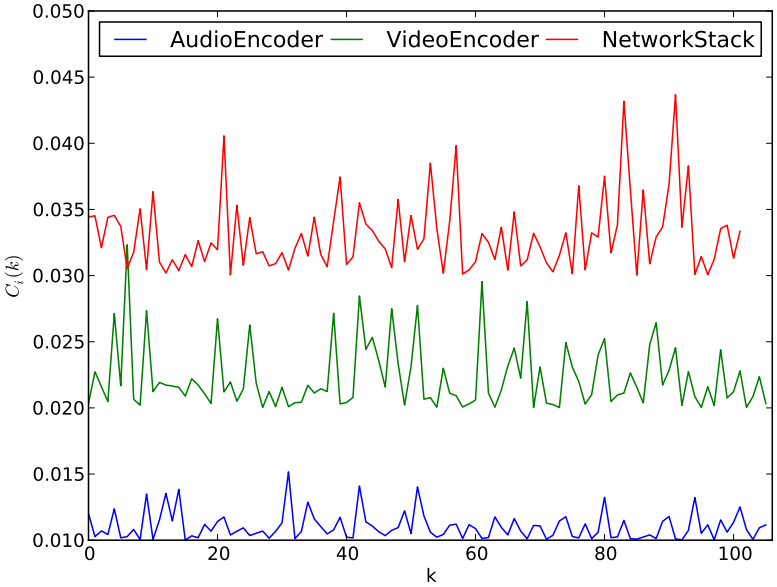


Figure 7.5 Generated cycle times used in the simulation examples below.

7.5 Simulation results

Simulation environment

In order to evaluate controls, a simulation environment has been developed in Python. The system dynamics is approximated by discretization with a time step of 1 ms, which incurs quantization on the cycle completion time stamps. This is, however, assumed to be of little effect as the variation due to noise is orders of magnitude greater for these simulations.

Cycle execution times have been generated as $D_i + X_i(k)$ where D_i is an a-priori unknown constant and $X_i(k) \in \exp(0.1D_i)$. The realizations used for the presented simulation results are shown in Figure 7.5. The randomness is meant to model both software execution time uncertainty and the stochastic properties of a modern CPU, including the effects of caches, the memory bandwidth gap and deep pipelines. The D_i values were chosen randomly so that the resulting k-parameters would lie between 10 and 100. A real sequence of cycle times for a video decoder is provided for comparison in Figure 7.6. h is globally defined as 1.

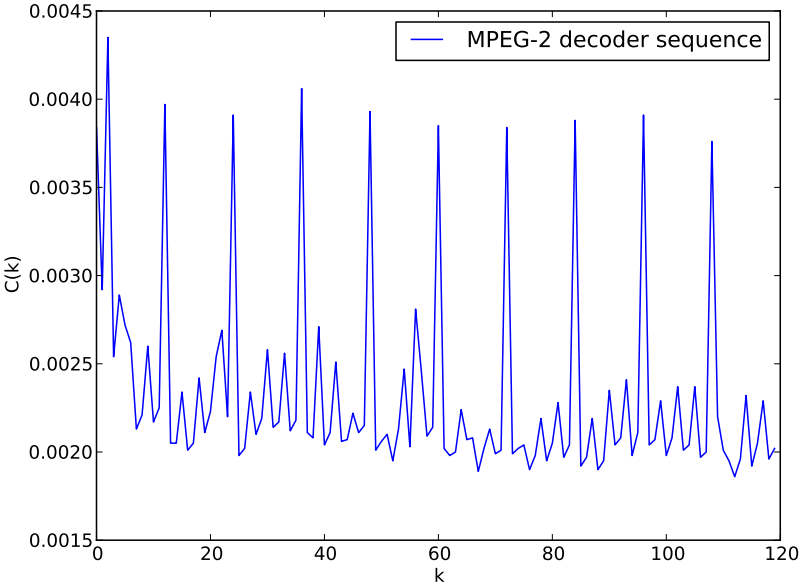


Figure 7.6 A sequence of execution times for an MPEG-2 decoder working on a video stream. This encoding standard interleaves complete image descriptions with delta descriptions. Decoding a complete image is significantly harder, causing the high peaks.

Thermal control

The thermal model of the CPU is based on [Fu et al., 2010b], but to make the effects of the thermal dynamics more visible in the simulations, the parameters have been scaled so the dynamics are faster. This makes the effects of control and disturbances more prominent. In the simulations, the parameters in Equation (5.8) are chosen as $a = 2$ and $b = 1.5$. The main purposes of the thermal model are

- to provide a scenario for the varying availability of CPU resource and
- to show how physical models can be combined with the software models,

so switching for more realistic parameters would not change the design decisions significantly though the thermal controller must then be re-tuned. However, as the focus of this thesis is on the application performance as affected by both hardware and software, more advanced temperature control strategies are left for future research.

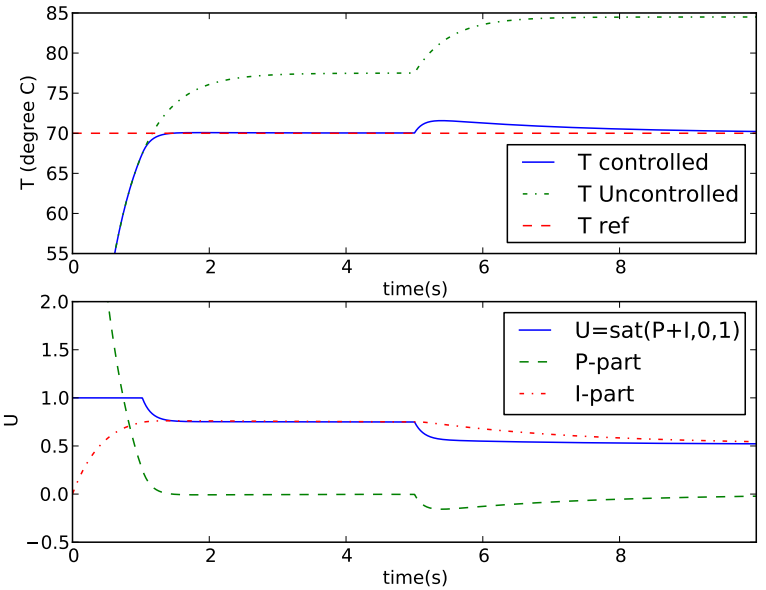


Figure 7.7 PI control of temperature with a constant disturbance of 15 degrees C / s entering at 5s. The uncontrolled dynamics are shown for comparison.

A scenario where a disturbance enters occurs at 5 seconds is shown in Figure 7.7. This could be a situation where the unit is left in direct sunlight that causes an insolation effect of 15 degrees C / s. The controller keeps the temperature by throttling the available CPU-time. As the temperature approaches the set point, the total utilization is lowered to about 80%. At 5 seconds, the disturbance causes the temperature to rise and the controller responds by lowering the utilization even further, settling at about 55%. If there are no active cooling measures or direct measurements of external disturbances, the set point must be sufficiently below the critical level to keep the CPU from overheating.

Parameter estimation

Figure 7.8 shows the estimated execution rates and corresponding k_i-parameter estimates over the same simulation. The discontinuous nature of the virtual flows is evident in these plots. Note that it takes some time before the network stack starts to execute and this is because of the queue-controller. It throttles the network stack while the queues are filled and because of this, the k-parameter estimator needs more time to form $\hat{k}_n$. This causes the large overshoot in the queue length before it settles on the desired level.

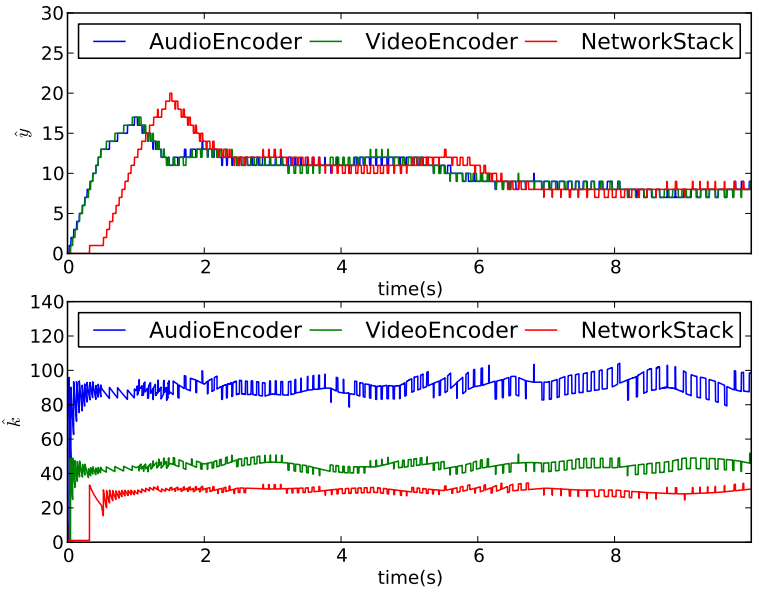


Figure 7.8 Rate and parameter estimation for all three tasks. Rates are estimated by counting events with a sliding time window with length 1 second. The network stack (red) lags behind in the beginning due to queue control.

Even though the actual execution rate changes over time, the estimate mean remains stable while the variance increases as the execution rates drop in the later part of the simulation, as seen in the upper plot. This is because a single event being outside or inside the estimation window will affect the estimate more. The window-based estimation scheme will break down when the rate drops below 1 cycle per second.

Latency performance

As there is no information about future demands for the CPU resource, a reasonable strategy is to minimize latency at all times. This is done by utilizing all available CPU-time while respecting the temperature set-point, thereby reducing the execution time for all tasks, and by controlling the queues. The reason why the queue controller is not trying to drive the queue lengths to zero is that this could cause blocking in the network stack, which in turn would reduce the accuracy of $\hat{k}_n$. This could be treated by designing a better estimator.

The latency control performance is displayed in Figure 7.9. A scenario without queue control is provided for comparison and the problem with this is evident. Even

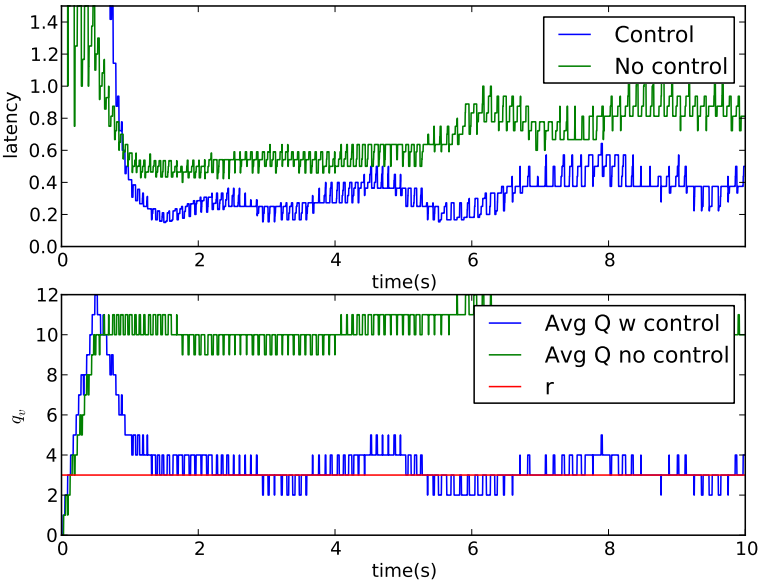


Figure 7.9 End-to-end latency and average queue length compared with and without control. The uncontrolled case (green) is about 2-3 times worse than what it obtained through control (blue).

though the system reaches steady state, where the queue lengths no longer change significantly, effects of the initial transient remains and cause significantly higher latency.

Sync performance

The simulations reveal the importance of the sync controller. Figure 7.10 shows that the sync error while left uncontrolled will actually drive the system to a stall. The reason for this is that the queue controller uses the average queue length to do the re-allocation. Figure 7.11 shows that even before the k-parameter estimates converge, the sync controller keeps the audio and video stream tightly together. This means that the queue lengths are actually the same, an assumption which can then be safely used by the queue controller.

Figure 7.12 shows how the queue lengths diverge quickly without sync control and the resulting re-allocation by the queue controller actually starves both audio end video encoder. It would theoretically be possible to form a MIMO-controller to handle both sync and queues in the same control law, but recall that the queue controller is a discrete time system that uses resource flow semantics and therefore

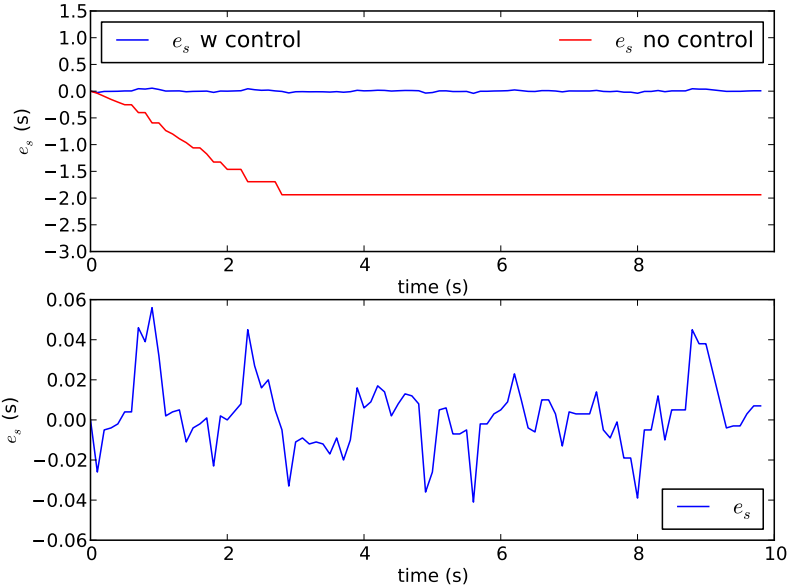


Figure 7.10 Sync error compared with (blue) and without (red) control. In the uncontrolled case, the encoding pipeline stalls which is why the encoding error seems to remain constant after 2.5 seconds. The lower plot shows the sync error from the upper plot in detail.

cannot in a simple way utilize information about individual events. The sync controller on the other hand operates on the event sequence and thereby has access to the cycle completion time stamps.

7.6 Thermal control through resource management

This section details experiments done on a mobile robot in order to validate the thermal control strategy through utilization throttling used in the above simulations. The experimental platform is a Pioneer 3, as seen in Figure 7.13 and further described in [MobileRobots, 2006]. The experiments are carried out using the ACTORS framework for resource management, but as the utilization bound has the same role as in the collaborative scheme from this chapter, the thermal control strategy is the same.

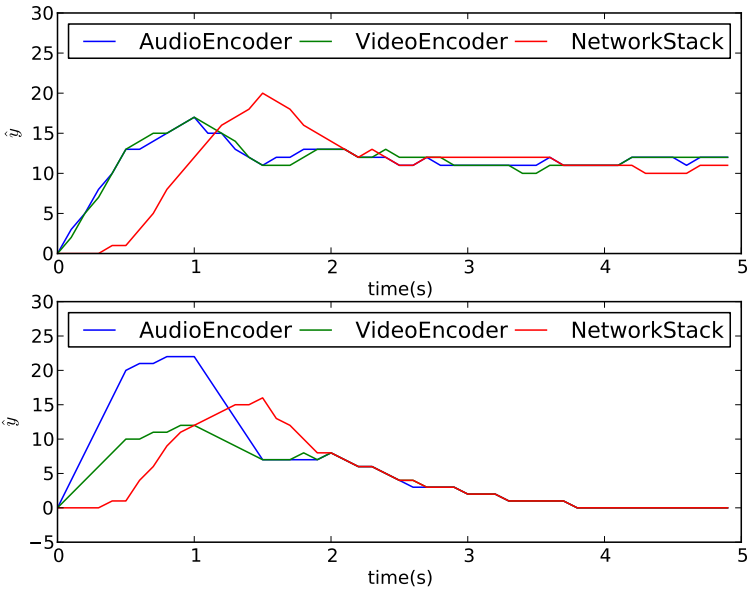


Figure 7.11 Execution rates compared with and without sync control. The curves have been averaged over a window of 0.1 s to provide better visibility.

System model

In order to prevent CPU processor overheating, which could cause performance degradation of all the applications executing on the system and even system failure, a system model that combines feedback and feedforward techniques to control both the temperature and the utilization of the processor (CPU) through adaptive bandwidth allocation is proposed. Figure 7.14 shows the proposed system model, which combines the features of a thermal controller, that keeps the temperature of the system bounded to a desirable temperature acceptable to the processor, and a resource manager that dynamically allocates resources to each application on the system. Here, T and T_R are the current temperature of the system, and the reference temperature respectively, it is assumed that this last value can be specified by the user. The values U, U_{min}, U_{max} and U_L correspond to the utilization of the system, the lower and upper utilization bounds and the utilization limit defined by the thermal controller respectively.

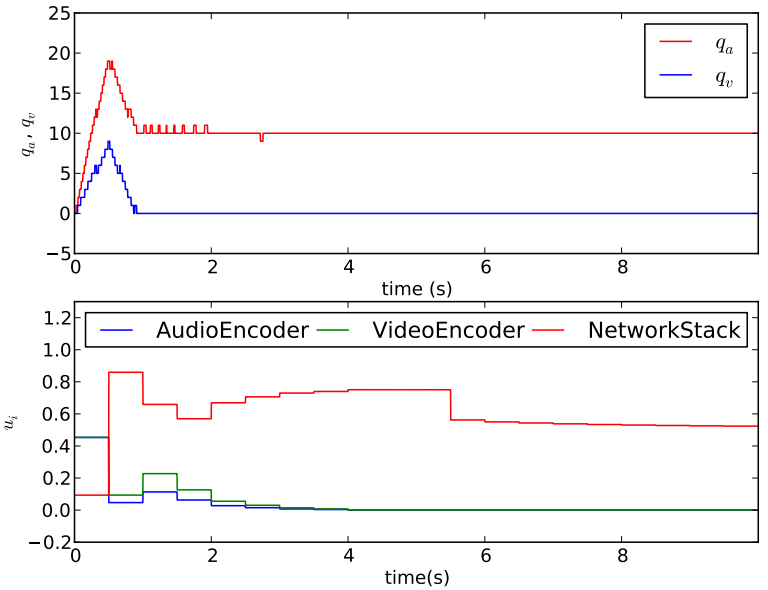


Figure 7.12 Queue lengths for a scenario with no sync control. The resulting allocation is based on the average queue length $(q_a + q_v)/2$.

Software components

Due to system limitations, as well as performance specifications, it is convenient to keep the temperature of the system bounded to a desired value. From a software point of view, this can be achieved using bandwidth reservation techniques [Abeni and Buttazzo, 1998a], which in combination with control theory allow at runtime adaptive allocation of CPU resources provided to the applications. Reservation techniques such as the constant-bandwidth server (CBS), guarantee to each application a certain execution budget every server period, this is also known as virtual processors (VP). In order to achieve this adaptive allocation of resources, a modified version of the architecture proposed originally by ACTORS (see e.g. [Segovia and Årzén, 2010]) is used, which assumes a multicore physical platform. For this particular case, a single core physical platform was employed. Figure 7.15 shows the modified architecture, which is composed mainly of three components: the applications, the resource manager (RM) and the reservation layer which includes the VPs of each application.

The RM is a daemon application, composed of a centralized supervisor and several bandwidth controllers, these elements will be explained in Section 7.7. The

Figure 7.13 The Pioneer 3 robot used for the experiments in Section 7.6, see [MobileRobots, 2006] for more information.

main tasks of the RM are to accept applications that want to execute on the system, to provide CPU resources to these applications, to monitor their behavior over time, and to dynamically change the resources provided according to the real needs of the application, and the performance criteria of the system, e.g., to limit the maximum temperature of the system.

The application can be composed of one or several tasks, which may have dependencies between each other. It is assumed that an application has different service levels. The quality-of-service (QoS) provided by the application is associated to the service level at which it executes, the higher the service level the higher the QoS, and the more resources it consumes over time, which implies a higher utilization of the system. It is also assumed that the application can communicate its service level information to the RM. Table 7.1 shows an example of this information for an application named A1 that has three service levels and three VPs, and for application A2 which has two service levels. In the table SL, QoS, α, Δ, and BWD are a service level index, the quality of service, the total bandwidth, the tolerable application delay, and the bandwidth distribution respectively. The delay Δ represents a measure of the time granularity of the specific service level, typically high

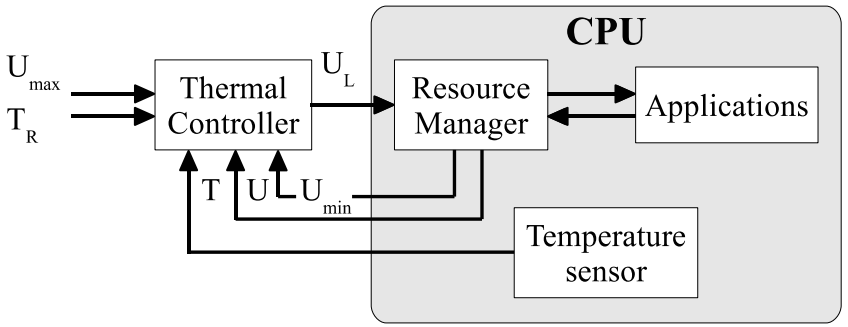


Figure 7.14 System model

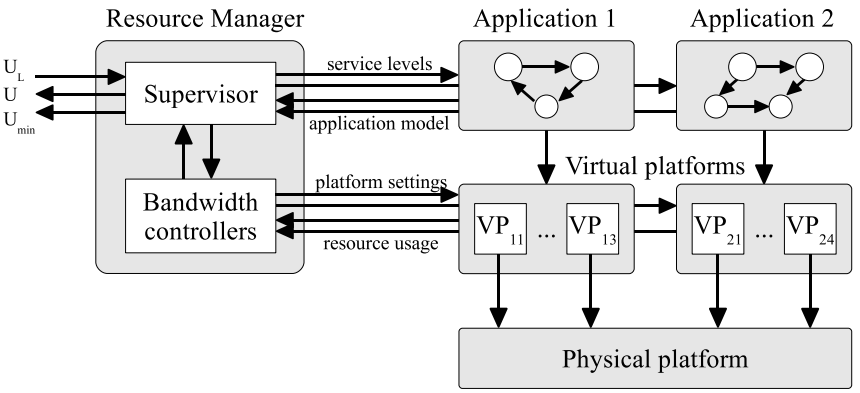


Figure 7.15 Modified ACTORS architecture

QoS levels have a low value of Δ. The bandwidth distribution is an optional value, it is an indication from the application to the RM how this total bandwidth should be distributed over the individual virtual processors. Additionally the RM is informed about the total number of VPs that each application contains and the importance of the application relative to others.

The information provided by the application (see Table 7.1) represents an initial estimate of the resources required at an specific service level. This information constitutes an initial model of the application, which during run-time and through the control mechanism implemented by the RM will be tuned appropriately.

Table 7.1 Service level table for application A1 and A2

Application name	SL	QoS [%]	α [%]	Δ [μs]	BWD [%]
A1	0	100	100	28-24-24	40-30-30
	1	80	70	42-48-48	30-20-20
	2	60	40	80-90-90	20-10-10
A2	0	100	60	20	
	1	80	40	50	

7.7 Control design

The proposed thermal control algorithm together with the resource manager are designed to meet two fundamental requirements: to prevent processor overheating by minimizing the maximum temperature of the system, and to provide desired system performance by maximizing the QoS provided by the running applications subject to the resource limitations.

Thermal control design

To fulfill the first objective of our control design, the use of a PI algorithm for the thermal controller is proposed. As shown in Figure 7.16, the signal T from

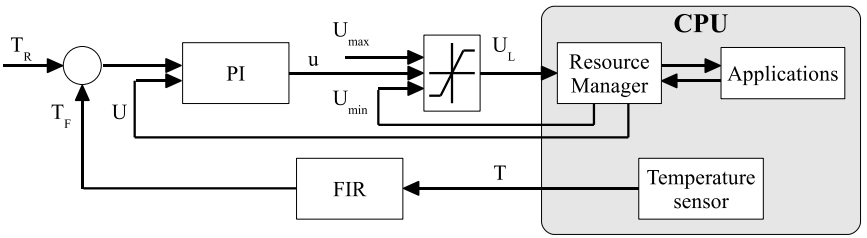


Figure 7.16 Thermal controller structure

the temperature sensor is passed through a FIR filter, which reduces measurement noise. The filtered temperature T_F is then compared with the reference temperature T_R, producing the error input of the PI controller. Additionally to this input the controller uses the utilization signal U provided by the resource manager, which represents the current CPU load caused by the applications running at an specific service level on the system. This signal is used by the anti-windup component of the PI controller to prevent wind up of the integral part.

The additional inputs U_{min} and U_{max}, represent the minimum utilization required by the applications to provide an specific QoS, and the maximal permissible utilization defined by the employed scheduling policy respectively. These two inputs provide the lower and upper limits that bound the PI controller output $u(k)$ such that,

$U_b \in [U_{min}(k), U_{max}]$. According to this the control output of the thermal controller, or utilization bound U_b, can be defined as $U_b(k) = sat(u(k), U_{min}(k), U_{max})$, with

$$sat(u(k), U_{min}(k), U_{max}) = \begin{cases} U_{min}(k), & u(k) < U_{min}(k) \\ U_{max}, & u(k) > U_{max} \\ u(k), & otherwise \end{cases}$$

The output of the thermal controller U_b, defines the input reference parameter for the resource manager, which based on this value, the measurements provided by the scheduler for each of the running applications, and the performance criteria of the system, e.g., maximization of the QoS, dynamically allocates CPU resources to the applications.

CPU resource allocation

The different elements that constitute the resource manager as shown in Figure 7.15, implement a control mechanism that combines feedforward and feedback strategies, which allow adaptive allocation of CPU resources at runtime.

The feedforward algorithm is carried out by the supervisor, which responsibilities include acceptance or registration of applications, monitoring of the minimum utilization U_{min} required by the applications to provide and specific QoS, and monitoring and control of the system utilization U, which is subject to the constraints defined by the thermal controller.

During registration, each application communicates its service level information (see Table 7.1) to the RM, in particular to the supervisor. Based on this information the supervisor assigns the service level at which each application must execute. This assignment can be formulated as an integer linear programming (ILP) optimization problem, which objective is to maximize the global QoS of the current applications running on the system including the new application, under the constraint that the total amount of resources is limited. The boolean variable y_{ij} is 1 if application i is assigned QoS level j, it is 0 otherwise. For each application i, q_{ij} denotes the quality at level j, and α_{ij} the bandwidth requirement. The problem can now be stated as follows

$$\begin{aligned} &\max \sum_i w_i \sum_j q_{ij} y_{ij} \\ &\sum_i \sum_j \alpha_{ij} y_{ij} \leq U_b \\ &\sum_j y_{ij} \leq 1 \qquad \forall i \end{aligned} \tag{7.22}$$

where U_b is the total assignable bandwidth of the system, and w_i is the weight (importance) of application i relative to other applications. The importance values are assumed to be decided by the system developer. The last constraint implies that,

if necessary, some low important applications might be turned off, in order to allow the registration of more important applications.

After the service level assignment of each application, the supervisor calculates the reservation parameters of each VP. Hence, it creates the VPs for the tasks of each application by defining the budget Q, and the period P of each VP. The calculation of the budget and the period of the server is based on the corresponding (α, Δ) (see [Mok et al., 2001a]) parameters described by the following equation

$$Q = \alpha P \qquad P = \frac{\Delta}{2} + Q \tag{7.23}$$

The service level assignment of the applications running on the system, is carried out not only during registration, but also when the thermal controller redefines the utilization limit U_b. This could be the result of abrupt temperature increments in the system caused by internal factors, such as a high computational load of the running applications, or by external ones such as overheated adjacent equipment. Any of these situations would trigger a new service level assignment for all applications.

The service level assigned to each application running on the system, sets an initial upper limit for the assigned budget also known as AB_L, this value can be directly calculated from the information provided by the application (see Table 7.1).

The feedback mechanism is implemented by the bandwidth controllers of the VPs of each application. They check, whether or not the tasks within the VPs make optimal use of the bandwidth provided, or the assigned budget (AB), and take actions to ensure this without degrading the performance of the application. The bandwidth controllers are executed periodically with a period that is a multiple of the period of the VP that they are controlling.

The bandwidth controllers measure the actual resource consumption using two measurements provided by the scheduler. The used budget (UB) is the average used budget over the sampling period of the controller at the current assigned service level. Considering that the linux scheduler `SCHED_EDF`, the name used by an early version of SCHED_DEADLINE, supports hard reservations, the UB is always less than or equal to the budget that has been assigned to the VP by the RM. The hard reservation (HR) is a value that tells the percentage of server periods over the last sampling period that the task in the VP consumed its full budget. This is an indicator of the number of deadlines missed.

The bandwidth controllers have a cascade structure shown in Figure 7.17. The HR_{SP} corresponds to the maximum percentage of deadlines misses that can be allowed in each sampling period. The controller C_1 defines the new value of the UB_{SP}, which in this case corresponds to the upper and lower bounds within which the UB should reside. In the case the bounds are violated, the controller C_2 either increases or decreases the assigned budget AB of the VP, this subject to the imitation defined by the supervisor. The value of HR_{SP} can be related to the performance of the application.

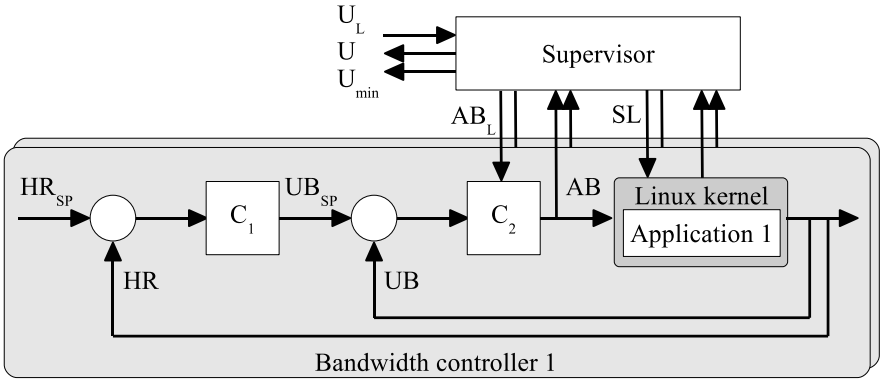


Figure 7.17 Resource manager controller structure

7.8 Implementation

The thermal control algorithm has been implemented together with the modified ACTORS framework. The implementation was done on a Pioneer mobile robot [MobileRobots, 2006] with an internal Intel Pentium III based computer [*Model VSBC-8 Reference manual* 2007]. The thermal sensor used is a National Semiconductor LM83 chip with an accuracy of $\pm 3°$C [National Semiconductor Corporation, 1999], and sample period of 2 seconds. The D/A-conversion takes approximately 500 ms and the temperature measurement is updated by the sensor driver every two seconds [Delvare, 2010]. In order to avoid aliasing effects, the sampling period of the bandwidth controllers is set to a multiple of the application granularity that is higher than the D/A conversion time.

A PI controller, designed as discussed in Section 7.7 is used to calculate the utilization limit parameter, that keeps the temperature of the system around a reference value provided by the user. To limit the measurement noise, the temperature signal is passed through a FIR filter with a rectangular window of one minute. The thermal controller is set to run as often as new data is available from the sensor, i.e., every two seconds, this is done to get as much data as possible to improve the filtering. The utilization bound U_b calculated by the controller defines one of the constraints of the service level assignment problem defined by equation 7.22. In order to solve the ILP optimization problem, the RM uses the GLPK linear programming toolkit [Makhorin, 2000].

The reservation mechanism is provided by the Linux scheduling class `SCHED_EDF`. The measured system utilization value only considers the applications that register with the resource manager, and not the RM itself which has a fixed amount of resources allocated by the system. For this implementation the maximum utilization U_{max} was set to 80%. Every time that the utilization limit

changes in any direction the system utilization will be temporarily higher, this occurs while the RM is solving the ILP optimization problem. To reduce this effect and to limit the influence of the noise that might still be present after filtering, the new calculated utilization limit is passed to the RM only if it has changed by more than five percentage points with respect to its previous value.

7.9 Experimental results

Thermal model validation and PI controller design

In order to validate the model structure presented by Equations (5.8) and (5.9), a step response experiment was carried out on the experimental platform. According

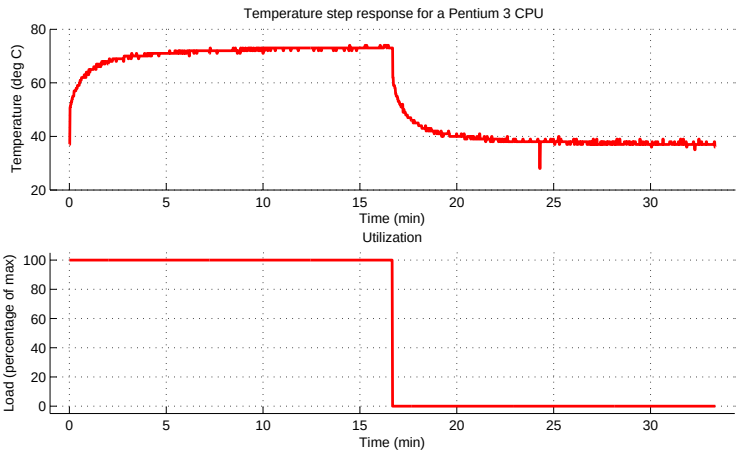


Figure 7.18 Step response experiment performed on a Pioneer mobile robot.

to the results shown in Figure 7.18, the dynamics between utilization U and chip temperature T can be roughly modeled by the first order system with time delay

$$\frac{T(s)}{U(s)} = \frac{K_P}{Ts+1} e^{-\tau s} \tag{7.24}$$

where the gain of the system corresponds to $K_P = 0.37$, the time constant to $T = 32.54\ s$ and the dead time to $\tau = 31.1\ s$.

According to this model, and the internal model control (IMC) approach, [Daniel E. Rivera and Skogestad, 1986], the tuning constants of the PI controller correspond to $K = 4.1792$ and $T_i = 48.09$.

Experimental setup

In order to see the performance of the proposed algorithm under normal and overloaded conditions, two different experiments were carried out. In the first experiment the reference temperature was set to 55°C for a period of 10 minutes, and then changed to 45°C for another 10 minutes. For the second experiment the reference temperature was kept constant at 50°C.

Since the objective of these experiments is to show the performance of the thermal controller working together with the resource manager, the service level tables of applications A1 and A2 are defined such that, the ILP optimization problem defined by Equation 7.22 always finds a feasible solution. The infeasible solution case which is handled by a bandwidth compression algorithm, and the tuning of the values on the service level tables are outside the scope of this thesis.

For the first experiment, a pipeline application A1 consisting of two tasks T_1^1 and T_2^1 with random execution times was used. As described in Section 7.7, during registration the application provides its service level information (see Table 7.2) to the RM. Since there are enough resources in the system, the RM assigns service level 0 to A1.

The first plot in Figure 7.19 shows the behavior of the filtered system temperature (green) with respect to the reference temperature (red). The second plot displays the measured utilization (green) and the utilization limit (red), which is the output of the thermal controller. The third plot shows the service levels of the application A1. The changes are done to compensate for the reference temperature change. The last plot of Figure 7.19 depicts the process variables of the bandwidth controller (see Figure 7.17), i.e., the used budget (green) and the hard reservation (blue) values, and the assigned budget (red) which is the output of the bandwidth controller.

As can be seen in Figure 7.19, after registration the bandwidth controllers read the HR and UB values of each of the application VPs and adjust them according to their set points, the HR_{SP} is defined as 0.1, that is, up to 10% deadline misses are allowed within each sampling period. During the entire execution of the application, the bandwidth controllers keep adapting the AB of the application. At time $t = 200s$ the execution time of the application decreases around 10%, the bandwidth controllers react reducing the assigned budget AB. A new execution variation can be seen at time $t = 300s$, this causes the HR value to 0.9, which is quickly compensated by the cascade controller.

Table 7.2 Service level table for applications A1

Application name	SL	QoS [%]	α [%]	Δ [ms]	BWD [%]
A1	0	100	60	24-32	40-20
	1	90	30	32-36	20-10
	2	75	20	36-36	10-10

Table 7.3 Service level table for applications A1 and A2

Application name	SL	QoS [%]	α [%]	Δ [ms]	BWD [%]
A1	0	100	40	28-36	30-10
	1	90	30	32-36	20-10
	2	75	20	36-36	10-10
A2	0	100	20	32	20
	1	85	10	72	10
	2	35	5	152	5

Since during the first 10 minutes the system temperature keeps below 55°C, the U_b value set by the thermal controller does not force a service level change in the application A1. At time $t = 600s$ the reference temperature changes to 45°C, here the thermal controller sets the U_b according to the algorithm described in Section 7.7. The changes in U_b trigger the feedforward mechanism of the RM, which assigns a new service level to A1. In order to compensate for the large temperature change, 3 service level changes are carried out, from 0 to 1, from 1 to 2 and finally from 2 to 1, where it remains. Notice that after time $t = 600s$ the filtered temperature T_F does not drop as rapidly as one could expect, this is the effect of solving the optimization problem that leads to a new service level assignment, and which increases momentarily the load on the system.

For the second experiment, a new pipeline application A1 and a simple application A2 consisting of one task T_1^2 was used, where application A1 has a higher importance than application A2. Table 7.3 shows the service level information provided by applications A1 and A2. At the beginning the only running application is A1, to which the RM assigns the service level 0. At time $t = 300s$ application A2 registers with the RM, which assigns service level 0 for A2 and keeps A1 at service level 0.

Figure 7.20 shows the behavior of both of the applications when it is required to keep the system temperature bounded to 50°C. This figure contains the same variables as described for the first experiment, together with the additional measurements corresponding to the second application A2. This can be seen specifically in the third plot, which shows the service levels for A1 (red) and A2 (green), and in the fifth plot which represents the measurement variables and the controller output of the bandwidth controller of the application A2.

When the application A2 registers with the RM, the utilization of the system increases causing an increment on the system temperature. Around time $t = 720s$ the thermal controller sets the utilization limit $u(k)$ to a value that requires a new service level change from the RM. This is carried out for both of the applications, but since application A1 has a higher importance than A2, the RM reduces the service level of A2 from 0 to 2. Once the system temperature gets below 50°C, the thermal controller increases the value of U_b, this causes a new service level

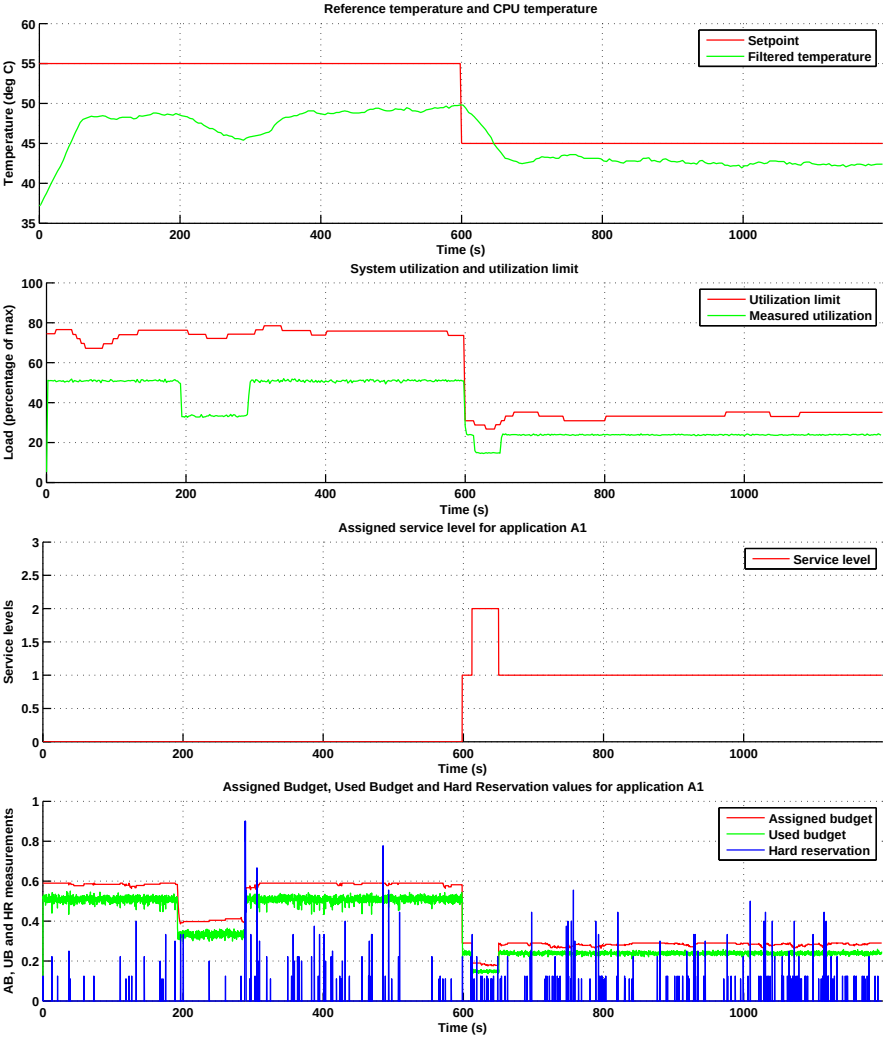


Figure 7.19 Performance results under normal conditions. One application running on the system subject to changes in the execution time, and in the system reference temperature.

assignment for application A2, from 2 to 1. The bandwidth controllers for both of the applications are also shown.

7.10 Conclusions

This chapter has presented a method for using feedback resource management to achieve thermal control on platforms with only passive cooling. Also presented are methods to control performance metrics for interconnected components under conditions of changing resource availability. Finally an experimental validation of using the thermal control approach on a mobile robot is presented, together with the performance results from an adaptive resource manager used on the robot platform.

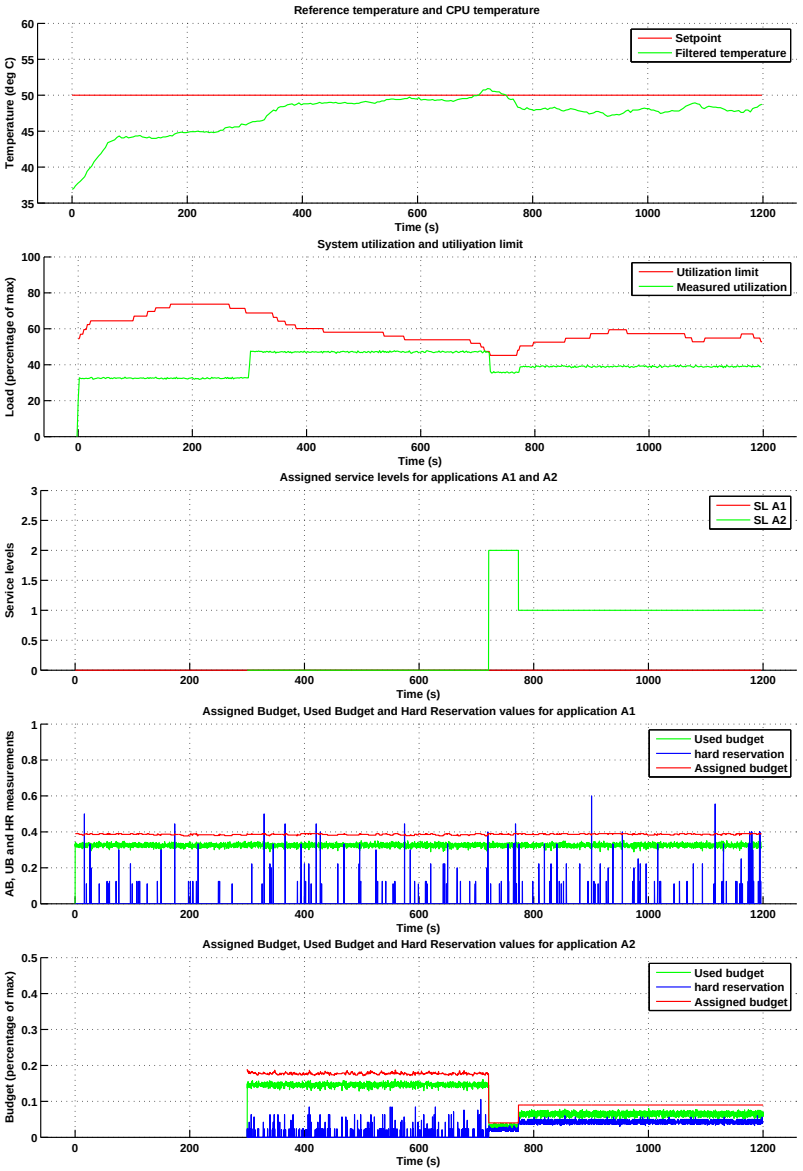
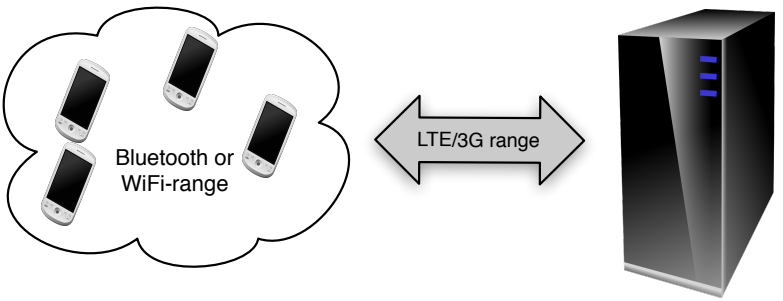


Figure 7.20 Performance results under overloaded conditions. Two applications running on the system subject to system temperature constraints.

8

Cooperative resource management



This chapter studies resource management of co-located energy-constrained devices performing individual but related tasks. Though ultimately seeking to improve their own performance, the devices are given incentive to cooperate, in this case sharing the energy cost of fetching data from a remote service, creating a win-win scenario for the individual and the system as a whole. Mechanisms affecting the resulting barter economy are investigated together with simulation results from both static and dynamic population cases. An asynchronous formulation is presented as a foundation for a prototype implementation. Experimental results from running the implementation in an emulated environment are presented along with design rules derived from the system performance. The chapter is based on the papers [Lindberg, 2013; Lindberg, 2014b; Lindberg, 2014a].

8.1 Increasing focus on the local

While the internet has enabled previously unprecedented connectivity between systems across the world, the rapid increase of data traffic is starting to be a problem for

the network infrastructure [FCC, 2010]. As a result, available wireless broadband spectrum is quickly being depleted, forcing a renewed focus on finding solutions that do not rely on limited infrastructure resources. This is also a relevant strategy when deploying solutions in previously undeveloped areas, e.g. rural Africa.

While battery capacity is steadily increasing, the desire to create both smaller and more powerful mobile systems continue to be a driver in mobile systems design. Technologies for very energy efficient short range networking are active research topics which receive significant attention from device manufacturers.

Mesh networking

IEEE 802.11s, a recent addition to the IEEE 802.11 family of standards, specifies a way that nodes can form an ad-hoc network and communicate with each other using a specialized routing protocol, the Hybrid Wireless Mesh Protocol (HWMP). Private communications between peers, or Mesh Stations as they are called in the standard, is conducted over encrypted links using a key exchange protocol called Simultaneous Authentication of Equals (SAE), which is also specifically designed to be used with IEEE 802.11s implementations.

Bluetooth Low Energy

Bluetooth Low Energy (BTLE, originally Nokia Wibree) is an addition to the Bluetooth Core Specification detailing enhanced features for short range communication, including device discovery and peer to peer connectivity. Support for BTLE is already present in the major mobile operating systems, including Apple iOS, Android and Windows Mobile, but the Bluetooth SIG is expecting the technology to also be used for more specialized devices, e.g. health monitors.

8.2 Incentivizing cooperation

It can be shown through game theory that two rational decision makers will choose not to cooperate if there is the possibility of one of them taking advantage of the other. This case is often referred to as the "Prisoner's Dilemma" [Axelrod, 2006] and illustrates the importance of trust between parties in a voluntary cooperation scenario.

Translated into the terms of a cooperative file retrieval scenario, if two geographically co-located mobile clients, A and B, seek to retrieve two data objects both clients want from a remote service (typically a file server), they could potentially cooperate by sharing the cost, in terms of energy or traffic fees, for the long distance traffic.

Let w_l denote the cost of accessing the data from a remote source and w_s the cost of communicating with a co-located source. Accessing data requires energy expenditure by both sender and receiver, meaning that one local access incurs a cost

Table 8.1 Action/reward table for the Prisoner's Dilemma description of the sharing scenario.

Cost for (A,B)	B cooperative	B uncooperative
A cooperative	$(w_l + 2w_s,$ $w_l + 2w_s)$	$(2w_l + w_s,$ $w_l + w_s)$
A uncooperative	$(w_l + w_s,$ $2w_l + w_s)$	$(2w_l, 2w_l)$

of w_s for both parties. The energy expended by the remote server is not accounted for in this example. Assume also that $w_l \gg w_s$.

A and B can now independently choose to either be cooperative, that is, allowing the other party to copy a data object, or uncooperative, that is, not giving the other client access. The outcome of the possible scenarios in terms of cost to retrieve both objects are listen in Table 8.1.

If both choose to cooperate, the costs will consist of one repository access to fetch one object and then two short range accesses, one to deliver the object to the peer and one to fetch the other object from the same. In case only one chooses to cooperate, both will start out fetching one object from the repository, but only one will be able to fetch from its peer. The other will then have to perform another repository access to complete the set.

The Nash equilibrium [Axelrod, 2006] of this game is that both A and B choose to be uncooperative. Two principle strategies can be seen that would resolve this problem, either

- w_s must be reduced to 0, thereby making it "free" to risk cooperation, or
- the off-diagonal choices must be eliminated, making it impossible for one party to exploit the other.

The first strategy models the behavior of traditional peer-to-peer networks, where sharing is considered to be without cost [Androutsellis-Theotokis et al., 2004]. Even if a participant is taken advantage of the majority of the time, the occasional win is achieved at no cost. In the mobile setting, the risk of running out of energy or loosing contact with surrounding parties makes such assumptions unrealistic.

It is therefore necessary that the exchange system provides potential participants with guarantees that the benefits outweigh the costs. In the general case, this would require appropriate models for user behavior, which is outside the scope of this thesis. A simpler scenario can be achieved by requiring that all transactions are bilaterally either cooperative or uncooperative, thereby effectively resorting to the second strategy above.

The question of whether or not to allow multicast transfers is problematic. While publications have shown that this improves the efficiency of cooperation [Wolfson

et al., 2007; Yaacoub et al., 2012], there is the risk that clients will opt out of the exchange system and just listen to multicasts, essentially re-introducing a Prisoner's Dilemma like situation. This work therefore assumes that all communication is unicast between two parties.

Furthermore, it is assumed that an exchange system can enforce the adherence to agreements between clients, which will in the case of data object exchanges be referred to as fair exchanges. Exactly how this is done is of less importance to the results presented, but for the sake of feasibility a prototype method is outlined below.

Prototype contract mechanism

Assume a set of parties C have agreed to exchange a set of objects D according to some scheme. The agreement, or contract, in the form of a list of tuples denoting (supplier, receiver, object) $\in C \times C \times D$, is handed over to an exchange system E, a physical 3rd party in the form of a remote service. E creates a set of encryption keys, one for each unique object in the agreement. The suppliers of each object are then given the corresponding keys, which they then use to encrypt the objects, after which they transmit them. When all receivers have signaled to E that they have indeed received the complete transmissions, E sends out the appropriate decryption keys to the participants.

8.3 System model

Consider a population of mobile terminals capable of multi-mode communication, that is, able to use several wireless communication standards. Specifically, they support both an expensive form of long range communication (e.g. 3G or LTE) and a less expensive short range alternative (e.g. WLAN or Bluetooth). Building on the notation introduced in Section 8.2, let $C = \{c_i, i = 1 \ldots N_c\}$ denote a subset of the population, such that all are within short range communication distance of each other. Furthermore, let $D = \{d_j, j = 1 \ldots N_d\}$ denote a set of data objects of uniform size that all clients desire to retrieve. These objects could be individual files or parts of one larger file, split into parts to facilitate distribution. While constraining the division to equally sized parts can seem like a simplification, it does help to make the market more efficient. If all parts require equal effort to exchange, finding suitable trade partners becomes easier.

All parts of D are available from a central repository accessible only over long range communication. The nominal cost for a client to download all parts of D is thus $N_d w_l$ but clients can cooperate by trading objects via short range communication and thereby reduce their total transfer cost. Because of the cost associated with sharing a data object with another client, it is assumed that a client will only provide a requested object if it is guaranteed to get an object in return, referred to as a *fair exchange*. This rule is referred to as the *exchange policy*.

The client population C, the target data set D, and the policy P under which trades take place constitute an Exchange System $E = (C, D, P)$, with the objective of allowing clients to minimize their data retrieval costs. The state of the system is the contents of the client side caches that contain the data objects once a client has retrieved it. When an object is exchanged between two clients it is copied, the original remains with the source.

In this formulation E implements a centralized decision mechanism with complete knowledge of the system state. The system dynamics evolve in discrete time steps of indeterminate length, but with the following logical sub-steps: *discovery, arbitration* and *effectuation*, that are repeated in a loop.

- **Discovery**. During discovery, clients join the exchange system and submit their current state. Let $\kappa(c)$ be a function that returns the data objects currently possessed by client c and $cardinal(\kappa(c))$ a function that returns the number of elements in $\kappa(c)$.
- **Arbitration**. Once the system state is established, the exchange system decides which trades that will occur. Clients not part of a trade will perform a default action, that can be either fetching an object from the central repository or passing (i.e., doing nothing).
- **Effectuation**. Finally all decisions are carried out. The completion of these actions marks the end of the time step, after which the next immediately starts with a new discovery phase.

Clients can be expected to join or leave the exchange system from one time step to the next, either voluntarily (e.g. having completed the data set or user command) or involuntarily (e.g. through loss of connection), making repeated discovery necessary. By choosing to pass in the arbitration phase, a client can bide its time, hoping for a more beneficial situation to arise in a future time step.

8.4 The dynamics of fair exchanges

The data object exchange scenario differs from many types of bartering economics primarily in that a client is typically interested in several trades rather than just one. It will withdraw from the exchange system once it completes its set of data objects. This removes an attractive cooperation partner, one who possesses all the objects desired by the other clients. This is a key complication that an effective exchange policy must take into account.

To keep the pool of cooperation partners as large as possible, it is important to prevent some from completing their sets far ahead of the rest. Making client state (i.e., the contents of the client side object caches) as diverse as possible will further increase the probability that any one client will find a peer that can provide desired objects, while needing those already in possession.

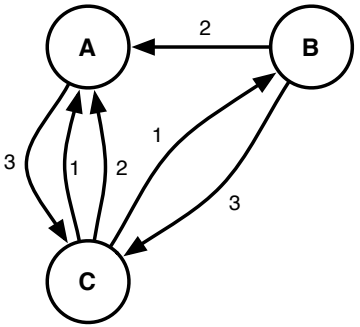


Figure 8.1 A swap graph for the system $A[1,2], B[1], C[3]$

The swap graph

To further discuss the properties of these systems, the concept of the swap graph will be used. This is a directed graph representation of currently possible trades, where each vertex represents a client and each edge represents a potential object transfer. If client A has an object desired by client B, then the swap graph will contain an edge from B to A, labeled with the object in question, to indicate the dependency. As there can be multiple dependencies between two clients, there can be multiple edges but with different labels.

As an example, consider a case with the client set $\{A,B,C\}$ and the data object set $\{1,2,3\}$. In the example, let client A possess objects 1 and 2, represented by the short hand notation $A[1,2]$. Assume now that the total system state is $A[1,2]$, $B[1]$, $C[3]$. The corresponding swap graph is seen in Figure 8.1. Possible fair exchanges are seen as cycles in the graph, with in total four in the example, as detailed in Figure 8.2. In this case they are mutually exclusive, which leads to the central question

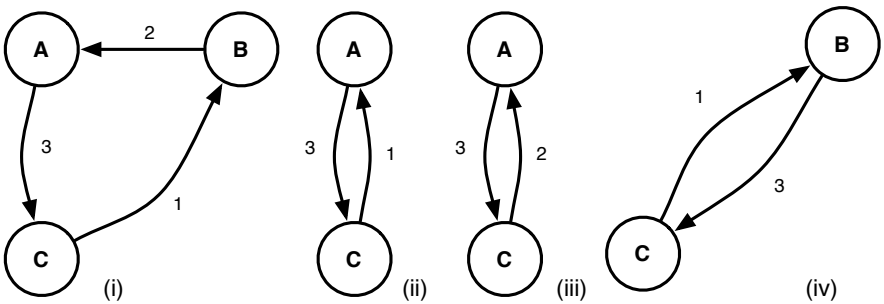


Figure 8.2 The cycles given by the swap graph in Figure 8.1. Each of the cycles is mutually exclusive, meaning only one of them can take place. Arbitrating this conflict is a responsibility of the exchange system.

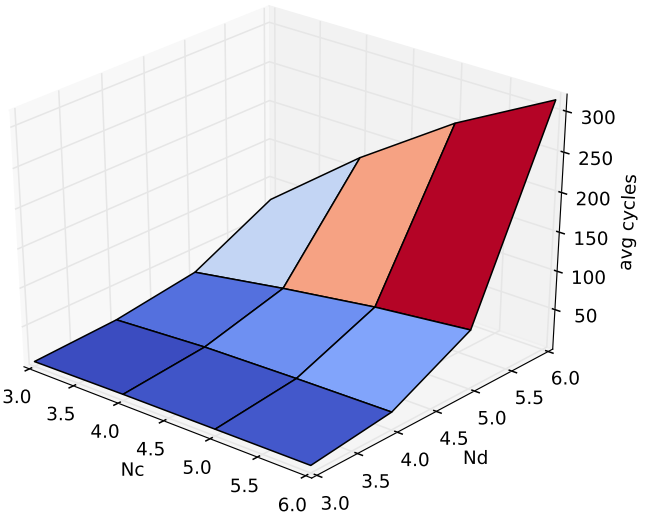


Figure 8.3 The number of cycles grows very fast with the dimensions of the system. This plot shows the average number of cycles over 50 simulations when all clients are assigned randomly chosen states.

of arbitration, that is, determining which exchanges should take place in order to optimize the objectives?

In its entirety, the problem is a multistep decision problem, where in each step determining the set of exchanges to perform involves finding the best set of non-overlapping cycles in the graph, a version of the classic NP-complete maximum set packing problem [Karp, 1972]. It has been shown how such problems can be solved through relaxation techniques after conversion to the maximum clique formulation [Ausiello et al., 1980; Balas et al., 1996], and this could theoretically provide a method to solve the complete problem, though the effects of the relaxation on this problem is currently unknown, as the goal is not the same as in maximum set-packing.

The computational complexity in each step grows as $2^{N_{cycles}}$ and since the number of cycles grows very fast with the set size, as shown in Figure 8.3, finding the globally optimal solution is intractable for realistic scenarios involving hundreds of clients. However, a possible key to alternative heuristic strategies presents itself by studying the special case of $N_c = N_d$.

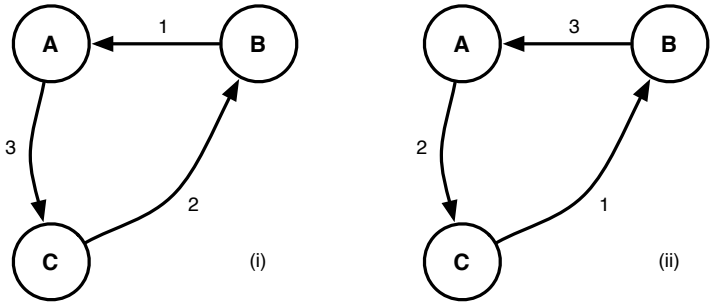


Figure 8.4 Swap graphs for the special case discussed in Section 8.4. The system is initialized with the state $A[1], B[2], C[3]$, as shown in Graph (i), which allows for a 3-way exchange involving all clients, leading to the situation depicted in Graph (ii).

The case of $\mathbf{N_d = N_c}$

Consider a case with $N_c = N_d = 3$, with the system state A[1], B[2], C[3] and the corresponding swap graph shown in Figure 8.4-i. After trivially selecting the exchanges, the state becomes A[1,3], B[1,2], C[2,3], with the swap graph in Figure 8.4-ii. This gives another trivial decision that ends the scenario (as all clients are done) with an optimal cost of exactly one remote access for all clients.

The two following observations can now be made:

1. The trivial optimal strategy above is always possible when all clients have the same number of objects and every object occurs the same number of times in the system.
2. The solution is not unique in general, there might be many other ways to achieve the same optimal global cost.

Let

$$n_c = \text{cardinal}(\kappa(c))$$

and

$$f_d = \sum_{c \in C} I_d(c)$$

where $I_d(c)$ is an indicator function defined as

$$I_d(c) = \begin{cases} 0 & \text{if } d \notin \kappa(c) \\ 1 & \text{if } d \in \kappa(c) \end{cases}$$

n_c is thus the number of objects possessed by c and f_d the number of clients possessing object d. Furthermore, let

$$\bar{n} = \frac{1}{N_c} \sum_{c \in C} n_c \text{ and } \bar{f} = \frac{1}{N_d} \sum_{d \in D} f_d$$

Using this notation, the condition from Observation 1 can be formalized into

$$n_i = n_j, \forall i, j \in C \text{ and } f_k = f_l, \forall k, l \in D \tag{8.1}$$

from here on referred to as Condition A.

Consider now the function

$$J = \sum_{c \in C} (n_c - \bar{n})^2 + \sum_{d \in D} (f_d - \bar{f})^2 \tag{8.2}$$

The quantity J can be seen to denote the distance to Condition A, or if it is assumed that the optimal trajectory will be followed once A is fulfilled, the distance to the optimal trajectory.

The quantities $\sum_{c \in C}(n_c - \bar{n})^2$ and $\sum_{d \in D}(f_d - \bar{f})^2$, essentially the sample variance of the client cache sizes and object frequencies respectively, can be interpreted to model two aspects of how well the exchange system will work.

If the cache size variance is high, then some clients will finish way ahead of others, thereby removing many objects from the system. It therefore makes sense to prioritize clients with few objects when arbitrating exchanges.

If the frequency variance is high, some objects are rare, meaning few clients can offer them, while some are frequent, meaning few clients want them. Both cases will lead to fewer possible exchanges involving these objects. It therefore makes sense to try to keep object frequencies uniform.

Analysis of J's impact on trading

To further demonstrate the correlation between the quantity J and number of trading opportunities, Figure 8.5 shows the result of a Monte-Carlo type simulation where a system state was generated 5000 times by randomly placing $N_d/2$ objects among N_c clients, for $N_d = N_c = 8$. For each random state, the swap graph was constructed and the number of cycles, i.e., the number of possible trades, were plotted against the corresponding value of J.

The negative impact on possible trades for high values of J is clear but for smaller values of J, the correlation grows weaker. It can therefore be expected that the algorithm presented in 8.5, which is based on this metric, will primarily work to prevent particularly bad trades, but that it will perform more or less on par with picking trades at random when the system is close to optimum.

8.5 Baseline algorithm

Using Equation (8.2), it is possible to formulate a one-step decision algorithm based on minimizing J. Basing decisions on only the currently measurable state of the system, in this case the contents of the client side caches, is a feedback control approach. This has the advantage of being robust to disturbances, such as failed transfers or clients arriving to or departing from the exchange system. A pre-calculated

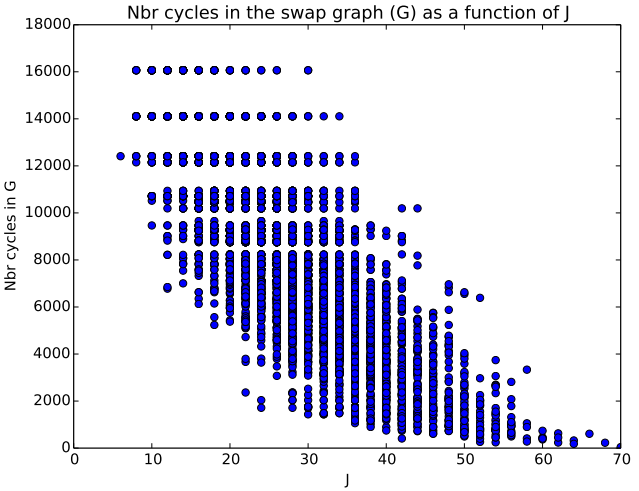


Figure 8.5 The correlation between J and the number of cycles in the swap graph G. Note how high values of J drastically limits the number of cycles, while the correlation grows weaker as J approaches 0.

multistep decision strategy would, on the contrary, have to be recalculated if, for instance, the state of the system suffers an unforeseen perturbation, such as a failed object transfer or clients leaving E.

Let X denote the system state, u denote a set of exchange agreements to carry out and $J(X|u)$ denote the cost function evaluated for the state after X has been subjected to u. Furthermore, let $\rho(X)$ be a function that maps the system state to a set of possible exchange agreements. The feedback arbitration policy can now be written as

$$u = \underset{u \in \rho(X)}{\arg\min}\, J(X|u) \tag{8.3}$$

Because of the combinatorial nature of the optimization problem used to calculate (8.3), designing the function $\rho()$ is non-trivial. The formulation is very close to the maximum set packing problem and as discussed in Section 8.4, the number of possible decisions grow unmanageably large even for modestly sized problems. However, it can still useful to compare other solvers with the result given if $\rho()$ is assumed to generate all possible agreements.

8.6 Heuristic solver

In order to further study this type of exchange system, a simple heuristic solver has been developed. Its main characteristics are that

- it is deterministic, that is, a given state always yields the same decision,
- it will always terminate (i.e., it is dead- and live-lock free),
- it is guaranteed to find at least one exchange unless there are none, which is trivial to test for, and
- it is computationally inexpensive.

Heuristic cycle finding

The heuristic solver uses a steepest decent style graph walking method for finding cycles. Let

$$J_c(X) = \frac{\partial J(X)}{\partial n_c} \tag{8.4}$$

$$J_d(X) = \frac{\partial J(X)}{\partial f_d} \tag{8.5}$$

and let G' be the *pruned swap graph*, obtained by repeatedly searching G for nodes with either zero in-degree or zero out-degree, i.e., nodes with either no inbound or no outbound edges, [Godsil et al., 2001], which obviously cannot be part of a cycle. As removing nodes from the graph can cause other nodes to qualify for pruning, the procedure is repeated until no more nodes can be removed.

As an example, consider the unpruned swap graph shown in Figure 8.6, defined on the data set $D = \{1,2,3,4\}$ and client set $C = \{A,B,C,D\}$. In the first step of pruning, client C would be removed as it has no inbound edges. Removing C in turn removes all inbound edges to A, meaning this node would be removed in the next pruning iteration. As the remaining nodes B and D have both inbound and outbound edges, no further reductions can be made and resulting graph is the pruned swap graph G'.

It can easily be seen that the all connected subgraphs in G' contain at least one cycle. As all nodes have outgoing edges with no edges to itself, there are no "dead ends" in the graph and since the graph is finite a walk must eventually end up in a node that has been visited before.

Because there can be multiple edges between nodes, the algorithm must keep track of both visited nodes and edges. This is done with a data structure called *trace*, a list of alternating node and edge elements.

A pseudocode representation of the algorithm used for finding cycles is given as Algorithm 1. `Sort()` orders elements in a list according to their unique text labels, which is done in order to make the algorithm deterministic. `FirstElement()` and

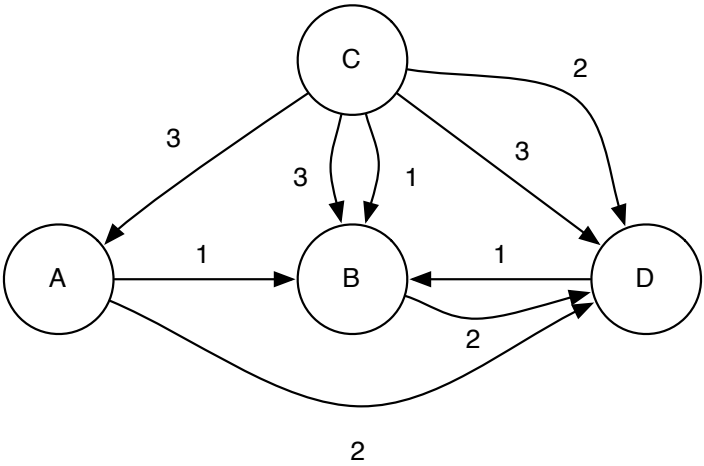


Figure 8.6 An unpruned swap graph on the data set $D = \{1,2,3,4\}$ and client set $C = \{A,B,C,D\}$. No edges are labeled 4 as all clients already possess this object.

`LastElement()` return the first and last elements of a list respectively and `List()` creates a list from the arguments. The function `gradJ` calculates $J_c(X) + J_d(X)$, using for c the end node of the edge and for d the data object associated with the edge. The last operation removes the preamble of the trace, as the initial parts might not be part of the cycle.

A heuristic $\rho()$

Building on the heuristic cycle finder, a heuristic $\rho()$, denoted $\rho_H()$, can then be defined. Let `HeuristicFind` be the heuristic cycle finder algorithm defined as Algorithm 1. A pseudocode representation of $\rho_H()$ is shown as Algorithm 2.

The exchange agreements to be carried out are generated by repeatedly searching for cycles in G and removing the nodes and associated edges in found cycles until the remaining graph is empty.

Computational complexity While the exact computational complexity of the algorithm is difficult to derive, there are ways to find approximate expressions.

The algorithm most often finds cycles of size 2. This comes from that the trades with the initial node have a high impact on the cost function, and the second node in the cycle will therefore often select it as a trade partner. Hence, it will take in the order of $N_c/2$ arbitration steps before the swap graph is empty. Each arbitration involves calculating the gradient of the cost function for all edges going out from the current node, move to the most beneficial one, and repeat until arriving at a previously visited node. Assuming cycles of size 2, this will happen twice for each

```
Input: pruned swap graph G'
Output: a cycle in G'
begin
    current ← FirstElement(Sort(GetNodes(G')))
    trace ← List(current)
    done ← false
    while not done do
        Es ← Sort(GetOutEdges(current))
        Gs ← List()
        mingrad ← MaxFloat()
        foreach e in Es do
            v ← SourceNode(e)
            d ← DataObject(e)
            u ← ReceiverNode(e)
            if GradJ(v, d, u) < mingrad then
                next ← v
                mingrad ← GradJ(v, d, u)
            end
        end
        append d to trace
        if next in trace then
            done ← true
        else
            current ← next
        end
        append next to trace
    end
    while FirstElement(trace) ≠ LastElement(trace) do
        remove first element from trace
    end
    return trace
end
```

Algorithm 1: Pseudocode representation of the heuristic cycle finder.

arbitration.

Figuring the number of edges from each node is more difficult, as this varies greatly depending on system state (see Figure 8.5). A theoretical upper bound for outbound edges for one node is $(N_c - 1)(N_d - 1)$, i.e., the client needs all objects but one and all other client have those objects. This has to be done once for each node in the cycle, which for the worst case is 2 nodes.

It can thus be concluded that the complexity of finding all trades in one arbitration phase is bounded by $N_c(N_c - 1)(N_d - 1)$.

```
Input: A swap graph G
Output: A list of traces representing exchange agreements
begin
    u ← List()
    done ← false
    while not done do
        G′ ← Prune(G)
        upart ← List()
        if not G′ empty then
            upart ← HeuristicFind(G′)
        else
            done ← true
        end
        remove nodes in upart from G
        append upart to u
    end
    return u
end
```

Algorithm 2: Pseudocode representation of the heuristic decision function $\rho_H()$ where `HeuristicFind()` is a call to Algorithm 1.

Default actions

The clients not part of any exchange are still able to act, though their actions are limited to either

- fetching an object from the remote repository, or
- passing (i.e., doing nothing).

The decision comes down to if the client is willing to wait and see if a better situation arises or if it should instead pay for immediate access to a data object. In this work, this is modeled by a client parameter named *trade timeout* that decides how long a client that is not part of any exchange agreement after arbitration is willing to wait, essentially a form of time out. An important special case arises when a client enters the exchange system with an empty cache. It will then have to download at least one object from the remote repository in order to have something to trade with.

As the other parts of the algorithm aim to minimize J, it would seem appropriate that the default actions do the same. This can be most easily done by simply fetching the least frequent object currently not already possessed by the client. In order to preserve determinism, the candidates are sorted by frequency first and label second.

Example: Empty initial state To further illustrate how the proposed mechanism works, here follows a step by step execution of the algorithm for a simple case. The

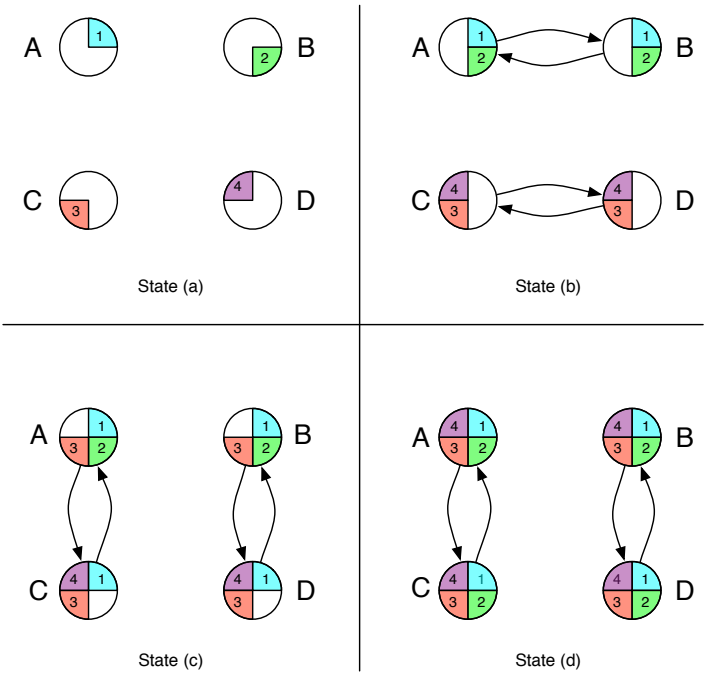


Figure 8.7 A graphical representation of the state changes used in the *Empty initial state* example. The example uses 4 clients ({A, B, C, D}) and 4 data objects ({1, 2, 3, 4}). The arrows represent the trades that resulted in the current state.

system state past the initial step is depicted in Figure 8.7. Consider a 4 x 4 system (4 clients {A, B, C, D} and 4 data objects {1, 2, 3, 4}) where all clients start out with empty caches.

1. Given the initial empty state, no trades can take place and all clients default to fetching an object from the repository. As they all pick the least common object, each will pick a different one. This results in State (a), as shown in the figure.

2. The cycle finder initiates by starting at the client with the least objects. Since in this case they all have the same number, it will simply pick A, which is the first according to sort order. This implementation sorts alphabetically after label. Since the gradient is equal in all directions, the cycle finder will go on to trade the first object A lacks with the first client that has it, in this case B. Continuing on in the same manner will lead back to A. As this forms a cycle, the cycle finder terminates and adds the trade between A and B to the list of decided on trades. A and B are then removed from the set of clients,

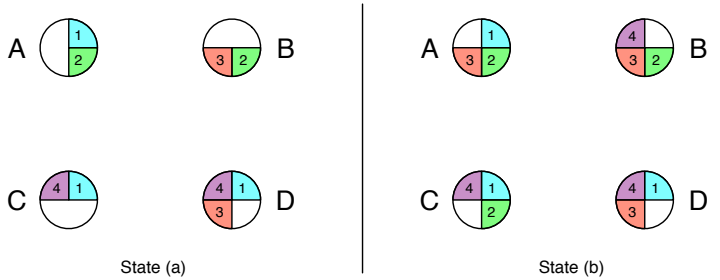


Figure 8.8 A graphical representation of the state changes used in the *Non-empty initial state* example. The example uses 4 clients ({A, B, C, D}) and 4 data objects ({1, 2, 3, 4}).

a new swap graph is generated and the procedure is repeated, finding a trade between the two remaining clients, C and D. After carrying out the trades, the system ends up in State (b).

3. Because the gradient is still the same in all directions, the system will continue to pick trades in sort order. This results in A trading with C and B trading with D, resulting in State (c).

4. The previous cycles will repeat themselves, resulting trades between the same clients and the scenario terminates in State (d), as all clients have all data objects.

The scenario illustrates how the system can work in an ideal case, where each client only needs to fetch one object from the remote server and then trade for the remaining.

Example: Non-empty initial state In order to show how the arbitration algorithm promotes certain trades over others, consider a 4 x 4 system configured in State (a) as shown in Figure 8.8, with client- and data sets as per the previous example.

1. The cycle finder starts with the first client with two objects, which is A. From there it picks the first edge with the lowest object frequency, which is 3, going to the first node with the lowest object count, which is B. The alternative would be to trade with D, but because D has more than the average amount of objects, the gradient is positive along this edge.

2. From B, the first edge with the lowest object frequency is 4 and the target with the lowest object count is C. Trades with object 1 are not preferred, since object 1 is more common than the average, and client D still has more than the average number of objects.

3. Finally, from C it picks the edge with object 2 leading to A, which completes the cycle. It could also go back to B, but will in this implementation pick A since it comes before B in the sort order.

As the trades are completed, D will, depending on the trade timeout, either default and download an object from the remote service, or wait until the next iteration. If D waits, the system reaches State (b) and the cost function J becomes 0. As such, the system will be able to find a solution where all clients can trade for their missing objects. This is exactly what the algorithm is supposed to do, i.e., promote trades that benefit the economy over time, which sometimes means some clients will be denied trades so that a better situation can develop.

A 10 x 10 example

The performance and behavior of the algorithm have been studied through simulations in an environment built in Python [Python, 2014] using off-the-shelf modules for graph algorithms [NetworkX, 2013] and plotting [Matplotlib, 2014].

Consider a scenario with 10 clients and 10 data objects, where all clients start out empty. Assume that $w_l = 1$ and that w_s is sufficiently small that it can be approximated to 0 under a policy of fair exchanges. Figure 8.9 shows how J, the total communication cost for the entire system and the worst case individual client communication cost evolve over time, stopping when all clients have all objects. Communication costs are normalized so that the nominal case where all objects are fetched from the remote repository corresponds to a cost of one. A plot showing how many clients are involved in trading in each step is also provided. The heuristic solver is used and clients use a *trade timeout* of 0.

As the initial state satisfies Condition A, the optimal trajectory is known and would give a worst case individual cost of one remote access, resulting in a normalized individual cost of $1/N_d = 0.1$, and a normalized total cost also of $1/N_d = 0.1$. The heuristic solver is not able to achieve these costs, but manages to reduce the total cost to 20 and the worst individual cost to 5, compared to 100 and 10 respectively for the non-cooperative case.

Influence of trade timeout

By varying the trade timeout parameter, it is possible to make a tradeoff between cost and latency. As expected, increasing the trade timeout generally decreases the resulting total cost, but finding the point after which increasing it further provides no benefit has so far only been done experimentally. Figure 8.10 shows a repetition of the setup from Section 8.6, but with a trade timeout of 2. In this case the optimal cost is nearly achieved, but the scenario takes more time steps, as can be seen in the last time step where the individual cost increases by one remote access. The total cost also increases but negligibly so.

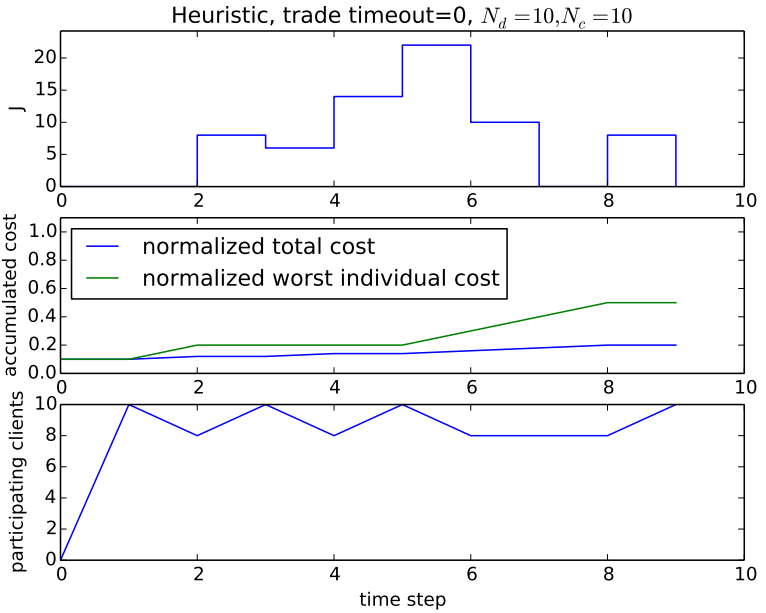


Figure 8.9 System trajectories for the example in Section 8.6. The costs have been normalized so that a cost of 1 corresponds to the worst case cost, that is the case where all objects are fetched from the remote repository. For the individual cost the normalization factor is $1/(N_d w_l)$ and for the total cost the factor is $1/(N_c N_d w_l)$. Without normalization, the accumulated total cost is 20, the average cost is 2 and the worst case cost is 5.

8.7 Set sizes and problem decomposition

The problem lends itself to decomposition into parts, as is evident when studying the results in Figure 8.11. This shows the worst case individual cost for different combinations of N_d and N_c (using the heuristic solver and a trade timeout of 10 in all cases) and it can be seen that the level jumps approximately each time N_d crosses a multiple of N_c.

This can be explained by treating the scenario as a combination of sub-scenarios.

Case 1: $\mathbf{N_d < N_c}$

If there are more clients than data objects, the clients can divide themselves into groups, ideally of size N_d, and apply the nominal strategy in parallel. Each client should only need to pay for one repository access.

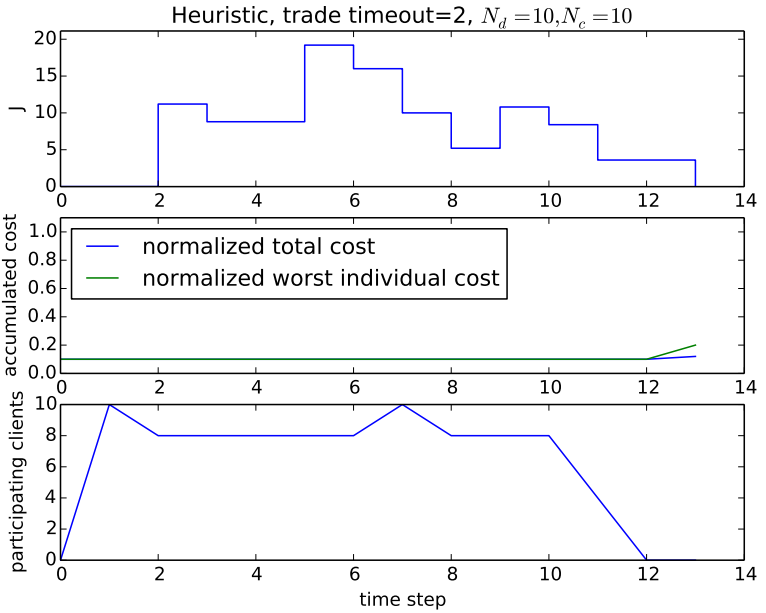


Figure 8.10 System trajectories for the example in Section 8.6, with the *trade timeout* parameter set to 2, resulting in lower individual and total costs at the expense of more time steps. Only a single client is forced to do two repository accesses, which is seen in the slight increase of worst case cost in the final time step. Without normalization, the accumulated total cost is 12, the average cost is 1.2 and the worst case cost is 2.

Case 2: $\mathbf{N_d > N_c}$

If there are more data objects than clients, then the data objects can be divided into groups, ideally of size N_c and handled in serial manner. Each client must pay for one repository access per serial group.

From this it can be concluded that

$$\lceil N_d / N_c \rceil \tag{8.6}$$

is a reasonable predictor for the worst case individual number of remote accesses using this solver or with words, a group of N_c clients can often cooperate around a set of N_d objects allowing each client to only pay for once remote repository access. If the remainder is non zero, some clients are likely to be prohibited from trading for one object, thereby raising the worst case number of accesses by one. Because of the above listed decomposition properties of the problem, (8.6) is also the lower bound on the worst case communication cost.

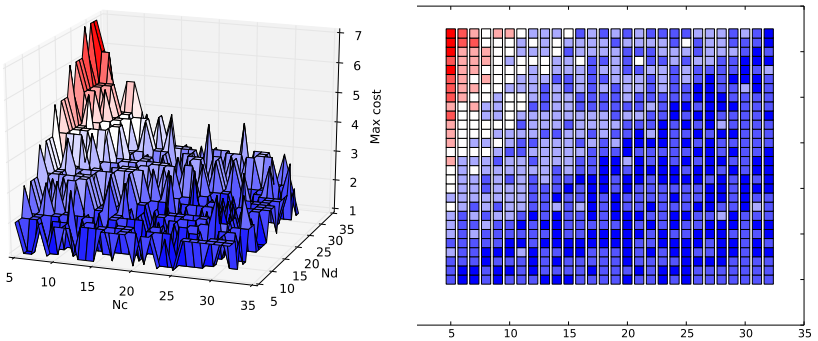


Figure 8.11 The worst case individual costs (non-normalized) for scenarios of various sizes, all using *trade timeout* of 10 which has experimentally been proven sufficient for all the cases in this simulation to reach their lowest costs. w_l is set to 1 (i.e., the cost is the same as the number of repository accesses), with the plot shown both from the side and from above to make the cost breakpoints more visible.

Variations on the cost function

It is natural to consider a more general structure on the cost function. Extending Equation (8.2) with the weights p and q gives the form

$$J = p \sum_{c \in C} (n_c - \bar{n})^2 + q \sum_{d \in D} (f_d - \bar{f})^2 \tag{8.7}$$

which can then be used to investigate the influence of the different parts. In order to show the effects clearly, a system with a skew object distribution state, both in terms of objects possessed by the clients and the frequency of the objects, will be used. For example, let the client set be $C = \{A, \ldots, J\}$ and the data set be $D = \{1, \ldots, 10\}$. Table 8.2 describes the initial state, where 'x' in position (i, j) signifies that client i has a copy of object j, that is, $\forall i, j;\ j \leq i \leq \max(i) - 2$. A set-up of this type will be referred to as a *skew 10x10* system. The reason the state matrix is not diagonal in the traditional way is that some clients would start with all objects already in possession and others so close to being finished that the influence of the cost function would be diminished.

Figures 8.12 and 8.13 shows the resulting costs for all clients to complete their data sets in skew 10x10 and 20x20 scenarios respectively, with the *trade timeout* set to 5. The balanced cost function with $(p,q) = (1,1)$ out-performs the alternatives except for the case where $(p,q) = (10,1)$ for the 10x10 example. The influence of the cost function is reduced for smaller problems, where the combinatorial nature of the problem is more dominant, and for cases where the *trade timeout* is very low.

A	x	x	x	x	x	x	x	x	.	.
B	x	x	x	x	x	x	x	.	.	.
C	x	x	x	x	x	x	.	.	.	.
D	x	x	x	x	x	.	.	.	.	.
E	x	x	x	x	.	.	.	.	.	.
F	x	x	x	.	.	.	.	.	.	.
G	x	x	.	.	.	.	.	.	.	.
H	x	.	.	.	.	.	.	.	.	.
I	.	.	.	.	.	.	.	.	.	.
J	.	.	.	.	.	.	.	.	.	.
	1	2	3	4	5	6	7	8	9	10

Table 8.2 A representation of the skew state used to examine the effect of the p and q parameters in the generalized cost function. An 'x' on position (i, j) signifies that client i has a copy of object j.

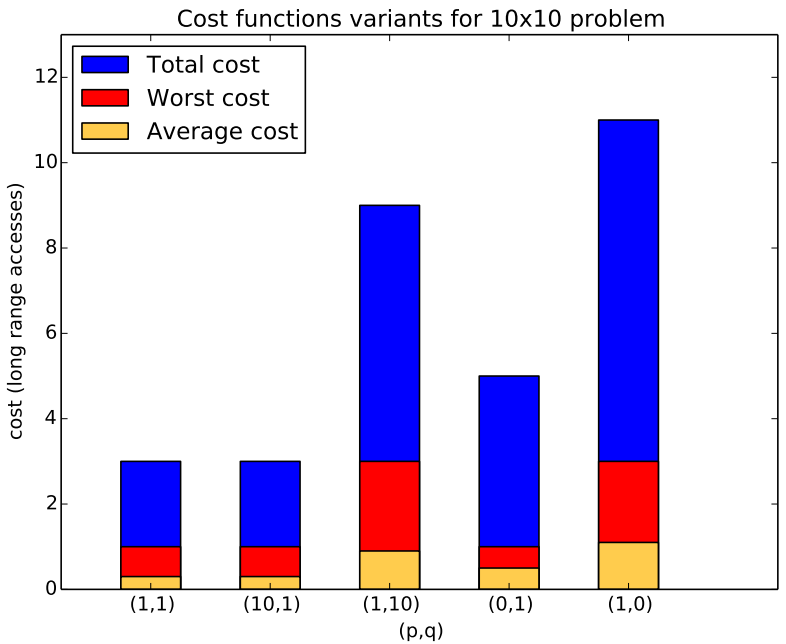


Figure 8.12 Resulting costs for different choices of (p, q) for skew 10x10 systems.

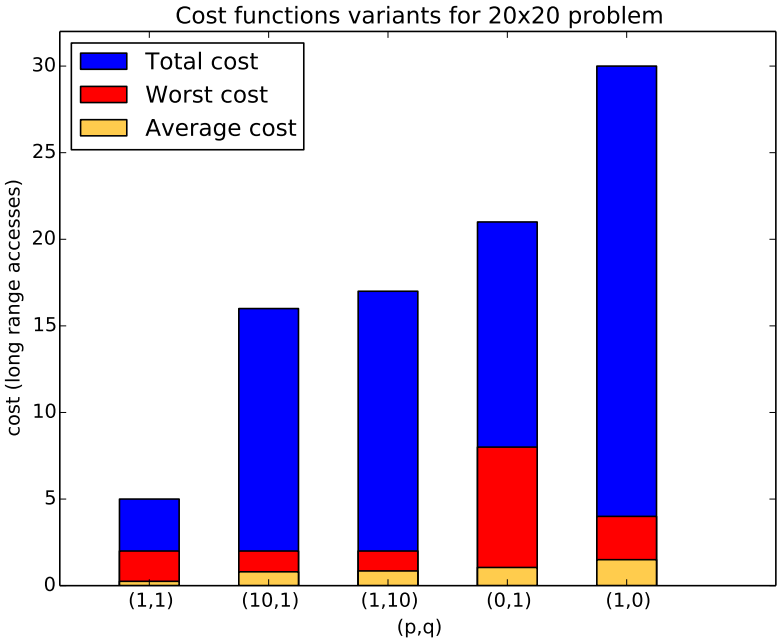


Figure 8.13 Resulting costs for different choices of (p,q) for skew 20x20 systems.

Comparison with a feed forward approach

The method presented in [Yaacoub et al., 2012] shows results with energy savings from approximately 60% (unicast case) to 95% (multicast case), when sufficiently many clients are involved. The results are not immediately comparable with those presented in this thesis, as the energy consumption of the IEEE 802.11b WiFi traffic is not sufficiently low to justify the assumption that $w_l \gg w_s$. However, the newer IEEE 802.11n standard has been shown to be in the order of a factor 10 times as efficient in J / bit as 802.11b [Halperin et al., 2012], making the assumption more realistic.

Using (8.6) as a predictor for the number of repository accesses, the cost to download N_d objects can be written as

$$\lceil N_d/N_c \rceil w_l + 2(N_d - \lceil N_d/N_c \rceil) w_s \tag{8.8}$$

Assuming w_s is low enough to be negligible and normalizing with the nominal cost of $N_d w_l$, the predicted normalized cost under the mechanism proposed in this thesis is $\lceil N_d/N_c \rceil / N_d$, meaning that given $N_c \geq N_d$, the cost scales with $1/N_d$. As such the predicted energy savings are on par with those presented in [Yaacoub et al., 2012], but note that those results would also be better if IEEE 802.11n had been used.

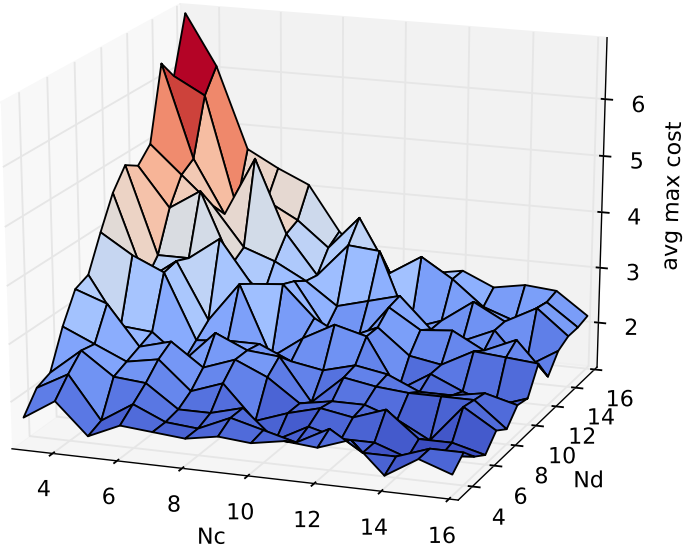


Figure 8.14 Average worst case individual cost over 20 simulations using a *trade timeout* of 10.

The main advantage of the feedback method in this thesis is that it does not require off-line optimization and therefore is able to handle uncertainties better, as show in Section 8.8.

8.8 Random initial state

Relaxing the assumption on the initial system state will give further insight into how the feedback based solver will handle a more realistic scenario. Clients might enter the system with some data objects already collected, others might join with empty caches at a later stage. Since the algorithm assumes all relevant information is part of the current system state, the exact events leading up to this point are irrelevant.

The result of a sequence of simulations, 20 for each (N_d, N_c) pair, Figure 8.14 shows the average max cost in a system with a random initial state, where each client possesses a random number of objects (uniformly distributed in $[0, N_d]$). The objects are in each case also picked at random, with uniform probability.

On average, a client in these simulations enters the system with half of the ob-

jects already in possession. It would therefore be reasonable to expect that the expected max cost would be lower than what the $\lceil N_d/N_c \rceil$ rule would predict, but the simulations suggest otherwise. The explanation for this is that some clients finish their sets early, thereby forcing others to pay for many repository accesses, which in turn increases the max costs.

8.9 Continuous operation

The approach presented in this thesis is primarily targeted at scenarios where the population will change over time, thereby making pre-calculated solutions unviable. A performance evaluation was therefore carried out using simulated scenarios for a case where $N_d = 16$ and with new clients arriving to the system through a Poisson process. The birth intensity was then made to increase over time to see how the system behaves under varying load. All clients in the simulation are modeled to have a trade timeout of 10, meaning they are willing to accept some latency in order to save energy.

Once a client has completed its data set, it will leave the system. The exchange system has a hard limit for the maximum number of clients it will accept and clients arriving when the system is full will simply be denied access and removed from the simulation. The limit could come from system resource constraints, such as memory, concerns about computational complexity and local network congestion or physical limitations, such as the maximum number of people that can reach each other with bluetooth or that can fit inside a vehicle with a local wireless network.

System capacity and congestion

Figure 8.15 shows how a system with a max capacity of 50 behaves under increasing arrival rate. The congestion point is very clear, the completion rate will not go above 3. The reason for this can be understood through the problem decomposition properties discussed in Section 8.7. As the average cost in this scenario is close to 1, the system performs nearly optimally. Therefore, the clients can be considered to group up in clusters of size N_d, where each cluster will complete its data set within N_d time steps. Each such cluster will therefore result in an average completion rate of 1 and with a maximum system capacity of 50, there can be $\lfloor 50/16 \rfloor = 3$ such clusters.

By increasing the maximum capacity of the system, more clusters can be formed and this will improve the throughput. Figure 8.16 shows how the completion rate will continue to match the birth intensity if the max capacity is increased to support at least 6 complete groups.

Throughput behavior for max capacity levels in between multiples of N_d remains difficult to predict. If $\lfloor N_c/N_d \rfloor N_d$ clients form complete groups, the remainder should be able to form fractional groups, as described in Section 8.7, thereby contributing with additional throughput. It seems, however, that the inability of the

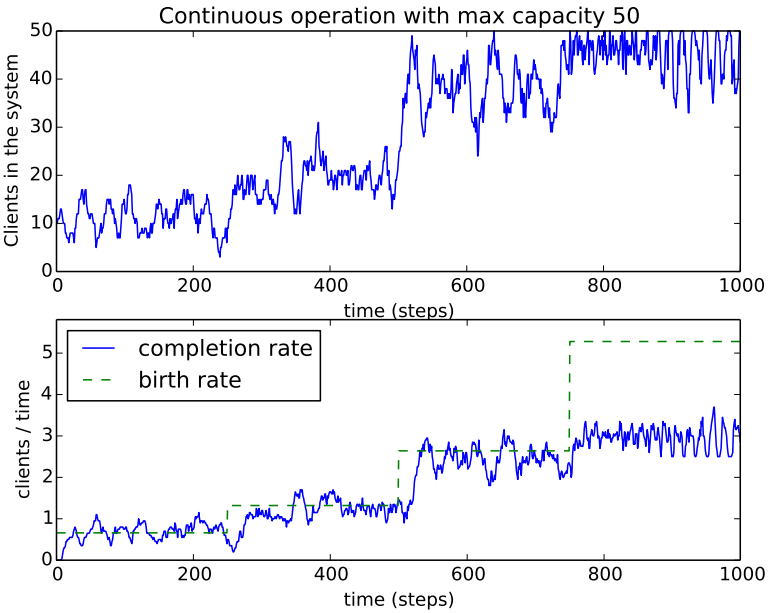


Figure 8.15 Simulation of an exchange system with $N_d = 16$ in continuous operation with new clients arriving through a Poisson process with increasing birth intensity. The system becomes congested when the intensity goes above 3. The completion rate is calculated over a 20 time steps long sliding time window.

arbitration algorithm to solve the optimization perfectly causes fluctuations in the throughput, and attempts to deduce this effect from the system parameters have so far been unsuccessful.

Returning to a max capacity of 50, Figure 8.17 shows what happens around the congestion point in more detail. In order to improve visibility, the birth process is in this case deterministic, i.e., a specific number of new clients arrive each time step. As before, the system becomes congested at a birth rate of 3 clients per time step. The oscillations in completion rate once the system becomes congested can be explained by considering the effect of the default client actions. Because newly arrived clients will fetch the least common object from the remote service in order to have something to trade with, a steady inflow of clients will keep the variance in object frequencies low. When clients are denied, the inflow of rare objects is reduced which forces clients to wait more often for beneficial trades, thereby reducing system throughput.

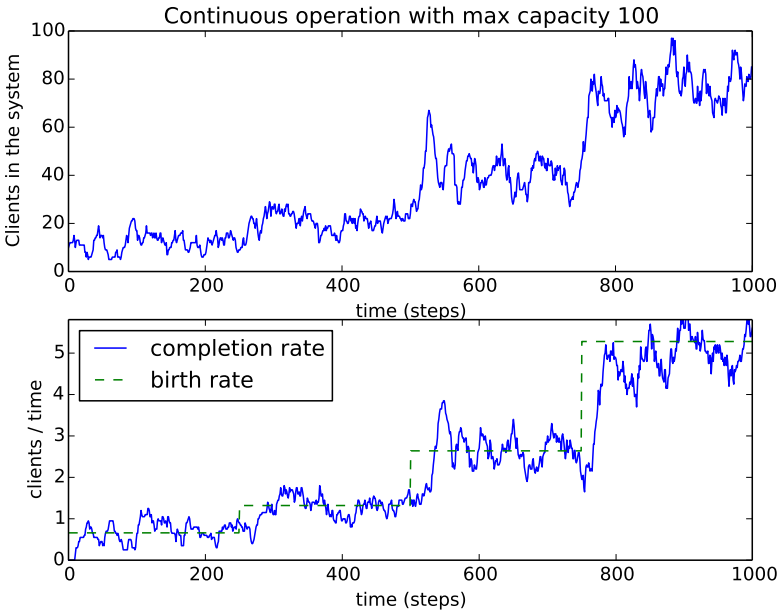


Figure 8.16 Simulation of an exchange system with $N_d = 16$ in continuous operation with new clients arriving through a Poisson process with increasing birth intensity. The system remains uncongested as intensity goes well above 3. The completion rate is calculated over a 20 time steps long sliding time window.

System design

Given this knowledge about when the system becomes congested, it is possible to relate some of the system design parameters to each other

$$t_{step} \geq \frac{S(D)}{rN_d}$$
$$t_c \leq t_{step} N_d$$
$$b_{max} \leq \lfloor \frac{N_{max}}{N_d} \rfloor$$
$$c_{tot} = w_l + 2(N_d - 1)w_s$$

where t_{step} is the length of a time step in seconds, $S(D)$ the size of the complete file in bits, r the lowest bit-rate used to transfer data either locally or remotely, t_c the time to collect the complete file, b_{max} the maximum arrival rate to be serviced, N_{max} the maximum number of clients in the system and c_{tot} the complete cost of collecting the file for each participating client.

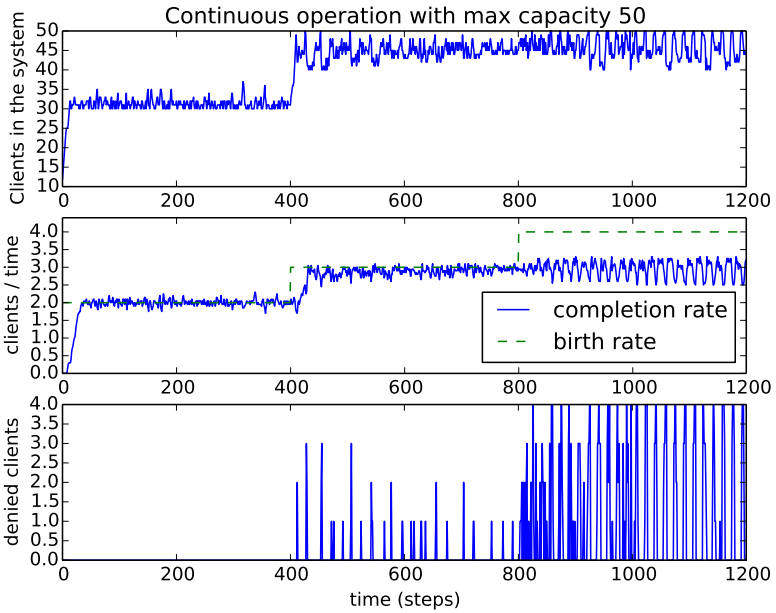


Figure 8.17 Simulation of an exchange system with $N_d = 16$ in continuous operation with new clients arriving at a deterministic rate that increases over time. The system becomes congested as intensity reaches 3, resulting in some clients being denied. As birth rates go even higher, the number of clients being denied increases, which reduces the inflow of rate objects to the economy causing fluctuations in completion rate. time window. Note that the max limit of clients is 50, the same as in Figure 8.15.

To illustrate how to utilize these design rules, consider a case where the objective is to dimension a system to be used in a commuter bus. Assume for this case that the iteration time is set to 1 minute and the bus has maximum capacity of 50 people of which 40 are assumed to be participating in the exchange system. Selecting how to divide a file, e.g. an operating system update, depends on minimum data rate, desired energy savings and how quickly the bus population changes. For maximum cost savings, the N_d should be as high as possible. However, this reduces system throughput. For maximum throughput, N_d should be as low as possible, but this reduces cost savings. If the passenger turnover in the bus is about 4 persons / minute, then $N_d \leq 10$.

8.10 Asynchronous formulation

The synchronous form used in Section 8.6 is not realistic to implement in an actual distributed system consisting of devices potentially very different from each other, as it relies on a system wide clock and full information available to the central decision mechanism. Therefore, this section introduces an asynchronous formulation that is used as basis for the implementation presented in Section 8.11.

A key difference is the introduction of *epochs*, here referring to a period in time where the decision server accepts connections from clients seeking to participate in the barter, further detailed below. Another significant change is the need to estimate the system state, i.e., how many clients are in the system and the contents of their caches, something that can potentially affect the effectiveness of the arbitration algorithm and the ability of clients to make decisions.

As there is no longer any global period, the *trade timeout* parameter used to model how latency tolerant a client is in the synchronous model is replaced by a *query timeout*, specifying how long a client is willing to wait for a trade opportunity before taking a default action.

Decision epochs

As one important assumption in this work is the dynamic nature of the client population, the central decision mechanism can no longer know exactly how many clients are present and what their intentions are. While the experiments carried out in this work uses a fixed client population, the specific clients that connect during each epoch varies.

As client connections are made, the server collects the incoming queries but will need to make decisions at some point in time to provide value. This point can be defined by a number of criteria, such as when a certain number of clients have connected, when the estimated system state is within some defined region, or when a predetermined time has passed.

This formulation uses the latter criteria, defining a decision epoch as a period in time of predetermined length, defined by the parameter *epoch timeout*. Connections occurring after the epoch timeout has occurred are sorted into the next epoch and so on. Once an epoch has been closed, the decision mechanism forms an estimate of the system state based on the client requests made during the epoch time span, as described below, decides which trades should occur, and then communicates out the results to the clients that are part of that epoch.

System state estimation

Because there is no way to learn the state of clients that have not connected to the decision server, the system must form an estimate as a basis for decisions. The two main strategies for doing this is to either assume that the clients that connected during the current epoch form the entire current population, or model the behavior of clients and use this to predict how the system state changes. The latter would need to

include a mobility model, describing how clients move in the physical world, which is beyond the scope of this thesis. It is also possible to formulate the state estimation as a consensus problem and piggyback state information on trades, something that will be necessary for a decentralized formulation.

The estimation technique used in this formulation is to use state information submitted by clients as they connect to the decision server and to also assume that they will act optimally, i.e., they will perform the trades decided for them and if they take a default action, they will choose to fetch the least frequent object, based on current system state estimates. The reason the system is designed in this way, rather than having the server explicitly tell the clients what default actions to take, is to make it possible for clients to make more strategic decisions in a future extension of this implementation. A client could for instance decide to continue to pass or change its *query timeout* based on the information given.

In the current implementation, the state is represented by a list of the labels of the data objects that the client currently has cached, e.g. [1,2,5,6,10]. The fact that the file is divided into a limited number of predefined parts simplifies the representation of the client state and calculations involving it. If arbitrary parts of the file would be tradable, not only would the server need to keep a much more detailed description of the contents in each client cache, but finding trades that would benefit all involved parties fairly would also be a much more complex procedure.

8.11 Software design

Using the asynchronous formulation from Section 8.10, a prototype implementation has been created and tested in a lab environment, as a proof of concept. The decision logic is implemented in Python, a practical high level language for describing complex algorithms, while the actual file transfer uses the standard SSH software distributed with most Linux-based operating systems, as this provides the encryption and access control necessary for securing the uni-cast only design while being a mature and efficient implementation.

Logically, the software consists of a threaded server application implemented using the Python 3 TCPSocketServer API [Python, 2014], and a client application also written in Python. The server uses the graph software package NetworkX [NetworkX, 2013] for part of the calculations. Figure 8.18 shows an overview of the hardware nodes and how the software components in the system are deployed.

Decision Server

The decision server implements the central decision mechanism and is the driving component in the content distribution system. It contains the original copies of the data objects that the clients seek to retrieve and handles the arbitration of conflicting trades. It also provides clients with estimates of object frequencies, which the clients then can use to make their decisions.

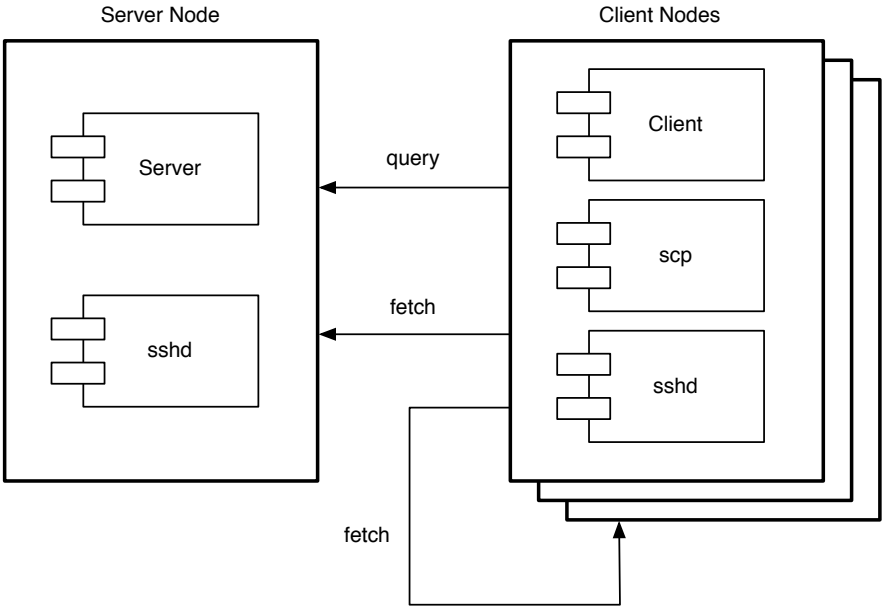


Figure 8.18 Component diagram showing the two node types and their software components. The Server and Client components are written in Python, while the sshd and scp components are part of the OpenSSH software distribution[OpenSSH, 2014].

The server operates in a primary loop where it listens to connections from clients and sorts them into epochs. When an epoch is closed, arbitration starts while new connections are accepted in parallel, as seen in Figure 8.19. An important design parameter is the epoch timeout, the choice of which depends on the use case. A large timeout can allow for many client connections, giving more complete information about system state and enabling more trade opportunities, while also incurring more latency and risking that clients disconnect and perform default actions. A short timeout can reduce latency, but increases the risk that not enough clients will sign into the system in order to facilitate trades, see Section 8.12.

Client implementation

The clients are relatively primitive, as the primary functionality is implemented using the SSH-server. The clients work in a single loop:

- First it queries the server, submitting information about the current contents of its cache in the process.

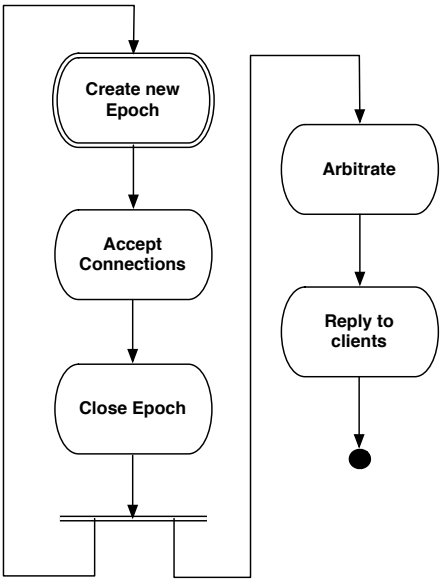


Figure 8.19 A diagram of how the main decision server loop operates. The system starts by creating a new epoch, which then accepts new client queries until it the timeout is reached and the epoch is closed. Immediately afterwards, a new epoch is created which accepts subsequent client connections, while the arbitration is carried out in parallel.

- The client then waits for the response until such is received or the client timeout occurs.
- If a trade offer is made, the client enables the trade partner to connect and then makes its own connection to fetch the suggested data.
- If no answer is given, or if the answer is empty, the client queries the server again for information about the object frequencies and then fetches least frequent one from the central repository.

Selecting the client *query timeout* parameter must be done with consideration of epoch timeout, but as is shown in Section 8.12, the effects of different choices can be non-intuitive as system behavior depends on the activity of all clients, see Section 8.12.

Energy model

Studies show that mobile phones use significantly different amounts of energy while transmitting over different network standards. For instance, using the models pre-

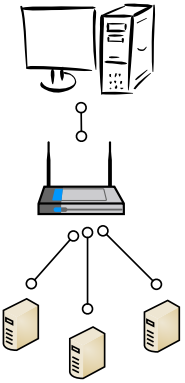


Figure 8.20 A depiction of the network topology used in the lab experimental setup, where the decision server is isolated on its own network segment while the clients share a segment.

sented in [Balasubramanian et al., 2009], the expressions for the transmission size to cost mapping for the energy case would be

$$T_{3G}(s) = 0.025s + 3.5$$
$$T_{WiFi}(s) = 0.007s + 5.9$$

for data objects s bits in size, indicating that given large enough data objects, clients could trade over WiFi and save energy compared to using 3G. For the purpose of this thesis, the costs will be simplified and w_s and w_l will be set to 0 and 1 respectively, meaning that the costs will be measured in the number of expensive accesses used.

8.12 Experimental results

The software has been deployed in a lab environment and run though different scenarios to see how the implementation of the asynchronous formulations performs. The decision server was populated with 10 data objects, considered to be wanted by all clients in the simulation.

Hardware setup and network topology

The hardware set-up used, consists of 13 Linux-based computers divided into two network segments connected through a switch, with the decision server located by itself to simulate it being accessed through a different network standard, e.g. LTE or 3G, as per Figure 8.20.

Example run, 5x10

To demonstrate how the system works, an example run with five clients, $\{A,B,C,D,E\}$ and 10 data objects is presented here. The clients start out with empty caches all activate at the same time. Figure 8.21 shows client actions from epoch to epoch, with '+' denoting a trade and 'o' denoting a default action. The server *epoch timeout* is 3 seconds and the client *query timeout* is 10 seconds.

As the experiment starts, all clients take a default action as they have nothing to trade with. They then reconnect and clients A-D trade objects with each other during the next epoch, while E is not offered a trade as the solver fails to find a cycle in the graph including it. At about 15 seconds into the experiment, all clients have the 5 objects fetched during the initial default actions, so no more trades can take place. However, as the client does not know that no more clients will connect, they wait, remaining passive until around the 24 second-mark when their *query timeouts* occur. This leads to another round of default actions that opens up for more trades.

The corresponding estimate of J is shown in Figure 8.22, together with an a-posteriori evaluation based on the client side logs. The latter is done by collecting the individual client logs after the experiment is done and parse the state evaluation. Note how the estimate is consistently higher than the calculated value, a phenomena that comes from that estimates are formed with no information from one or more clients, or old information in case queries have timed out and the clients had taken a default action. A spike in estimate error occurs around 25 seconds, at a moment when many of the clients have timed out and taken default actions. The server is then forced to form the state estimate without recent information from many of the clients.

Example run, 12x10, long query timeout

The scenario is similar when scaling up to 12 clients, though because there are now at least as many clients as there are objects, the mid-scenario pause seen in the previous example is not present, as can be seen in Figure 8.23. For this experiment, all computers in the lab setup were used, making $N_c = 12$. The pause is eliminated because all objects are actually possessed by at least one of the clients once the initial fetch is completed. The long timeout gives the system ample time to find trades, which in this case results in no extra default actions taken. As the client decisions are in this case completely timed by the server, the system essentially behaves as synchronized. Since no client reach the *query timeout*, there is no point in investigating even higher values.

Example run, 12x10, medium timeout

Reducing the time out is one way to avoid long breaks in case objects are missing in the population. However, as shown in Figure 8.24, this can result in fewer client being present for arbitration as they are taking default actions, reducing the overall performance of the trade economy. The *query timeout* used here is 5 seconds.

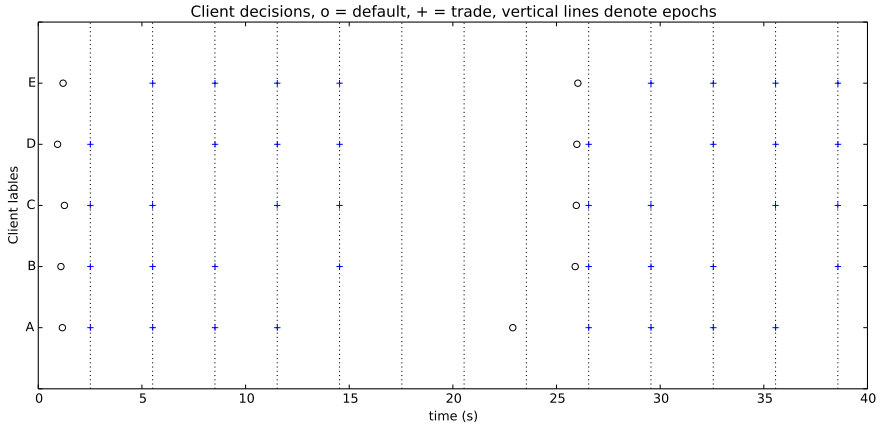


Figure 8.21 A graphical representation of what actions the clients are taking over the course of the experiment. '+' denotes a trade while 'o' denotes a default action. The vertical lines show the epoch timeouts.

Example run, 12x10, short query timeout

Using even shorter timeouts intuitively results in more default actions, as shown in Figure 8.25. In this case, the *query timeout* is 3 seconds, the same as the server epoch timeout. In this example, the short timeouts lead to a significant increase in default actions being taken for some clients (e.g. K and L). Using an even shorter *query timeout* is questionable, as this may not give the server time to perform the arbitration.

Effects of client timeouts

For a client seeking to optimize either latency or energy savings, picking an appropriate *query timeout* is important. The shorter the timeout, the shorter time the client will be waiting for a trade opportunity. However, because trade opportunities are most easily found when many clients with diverse cache contents are available, signing in to the trade system at the same time as the rest of the population turns out to often be more significant than the timeout.

Looking at the cost saving part of the experiment, Figure 8.26 shows the worst and best number of expensive long range accesses used by clients for a some different choices of *query timeouts* in scenarios with 10 clients and 12 data objects. As expected, using long timeouts will likely enable clients to trade for most objects after the initial default action has been taken, while at lower timeouts, the variance in savings increases.

Table 8.3 shows the time it takes for clients to complete their entire data set given different values on their *query timeouts*. In all cases, the epoch timeout is

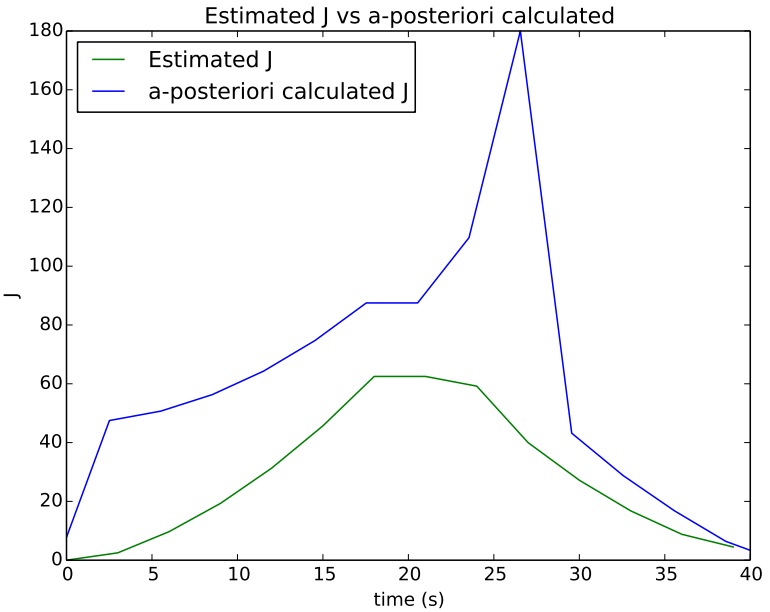


Figure 8.22 *J* as a function of the server side system state estimate, sampled at the end of each epoch, compared to an a-posteriori calculated value based on the client side logs. Note how the estimate is always higher than the actual value, but the trend is generally the same.

3 seconds. Non-intuitively, shorter timeouts does not seems to be correlated with shorter completion times. This can be explained by considering that a system with very long query timeouts will be almost entirely driven by server decisions, provided the system is not close to being empty. This causes the clients to behave almost in a synchronized manner (see Figure 8.21), which is beneficial to the trade arbitration.

When the client timeouts are shorter, they will more often act out of synch with the rest of the population, causing something similar to a traffic congestion on freeways [Treiber et al., 2000]. Interestingly, the worst choice seems to be a timeout larger than the epoch timeout, but not significantly larger. Using a long timeout is, however, not without risks, as a client could arrive to a system with few other connected clients or clients with few desirable data objects. In those situations, it is less likely that the decision server will find trading opportunities for all connecting clients each epoch, leading to long wait times but yielding no savings.

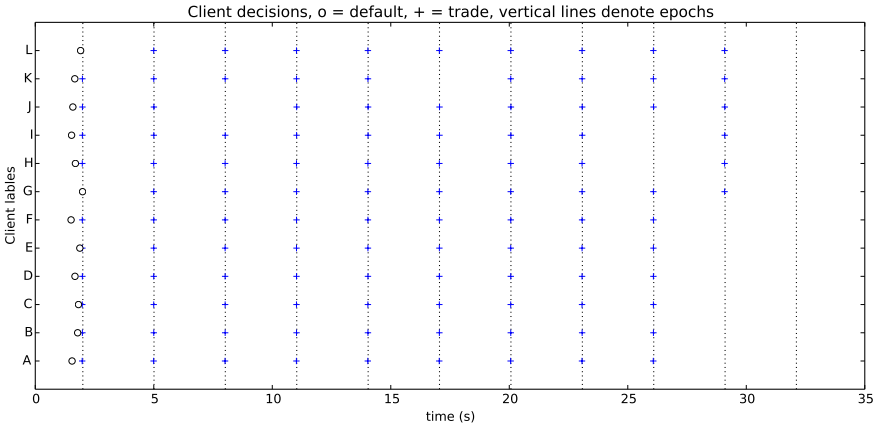


Figure 8.23 A graphical representation of what actions the clients are taking over the course of the experiment for a case where the *query timeout* is 10 seconds. '+' denotes a trade while 'o' denotes a default action. The vertical lines show the epoch timeouts.

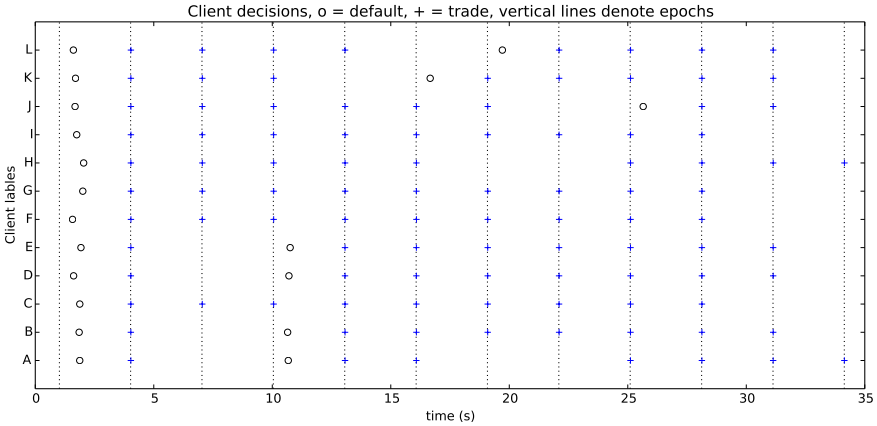


Figure 8.24 A graphical representation of what actions the clients are taking over the course of the experiment for a case where the *query timeout* is 5 seconds. '+' denotes a trade while 'o' denotes a default action. The vertical lines show the epoch timeouts.

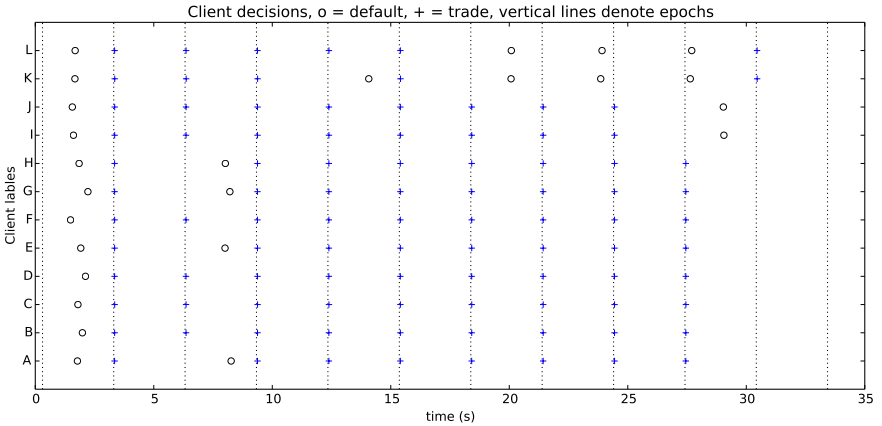


Figure 8.25 A graphical representation of what actions the clients are taking over the course of the experiment for a case where the *query timeout* is 3 seconds. '+' denotes a trade while 'o' denotes a default action. The vertical lines show the epoch timeouts.

Table 8.3 Completion times, i.e., time from client activation to data set completion, for experiments run with different *query timeouts*. The variance is largest for the 5 second timeout and interestingly enough the 10 second timeout performs better than the 3 second timeout does.

Client	10 s	5 s	3 s
A	26.86	40.72	28.27
B	26.88	31.97	28.27
C	26.86	28.90	28.25
D	26.92	31.98	28.30
E	26.90	31.99	28.25
F	26.91	28.91	28.26
G	29.93	28.99	28.29
H	29.91	34.94	28.25
I	29.93	28.94	29.05
J	29.94	31.93	29.03
K	29.91	31.99	31.28
L	29.87	31.99	31.31

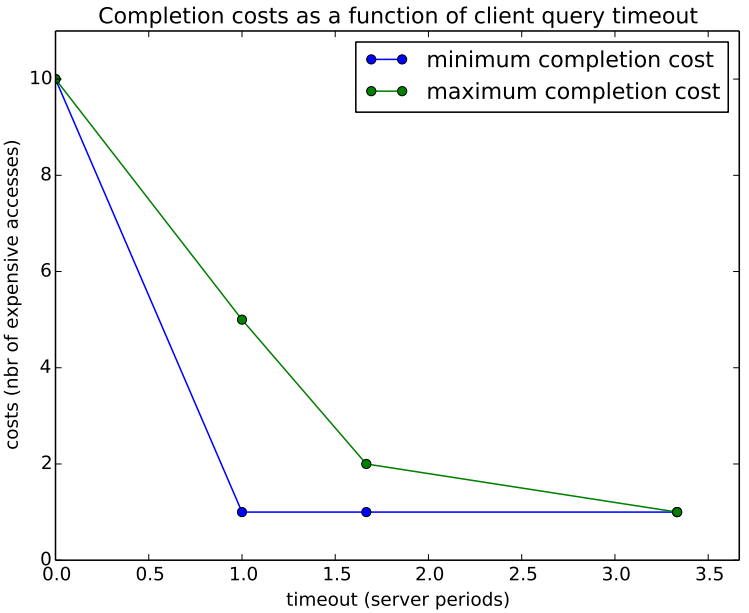


Figure 8.26 The maximum and minimum costs for a client to complete its data set as a function of the *query timeout* setting. For high timeout values the performance is consistent while for lower values it starts to fluctuate. A timeout of 0 means that the client always performs a default action.

8.13 Conclusions

This chapter has presented an approach for cooperative content distribution for mobile networks with the intent of conserving energy and long range spectrum resources. The approach is based on a mechanism where mobile clients trade data in a barter economy-like manner. Trades are arbitrated using a feedback control approach, using a cost function shown to correlate to the efficiency of the economy.

The approach is presented in closed population as well as dynamic population scenarios and an experimental validation through an implementation in a lab environment is supplied.

9

Conclusions and future work

9.1 Conclusions

This thesis presents three different perspectives on feedback-based resource management for cyber-physical systems. The initial viewpoint is the competitive scenario, when unrelated components share the same limited resource. The second viewpoint is the collaborative scenario, where components contribute to the same performance metric while sharing a limited and time varying resource. The third and final viewpoint is the cooperative scenario, where unrelated clients choose to cooperate around a task to better utilize limited resources.

All scenarios are characterized by limited knowledge of component dynamics and resource availability, making feedback-based methods an attractive alternative to overly pessimistic worst case designs. Online event-based estimation techniques are used to reduce uncertainty and the need for software designers to provide detailed information about resource needs.

Convex optimization is used as a mechanism for online decision making in the competitive case, while taking care not to expend too much resources on solving the optimization problem. The feedback-based approach performs worse under conditions with perfect knowledge, as some amount of resources are unavoidably used up by the decision making system and the error when estimating system parameters will further reduce performance.

Designing for feedback

The software models used in this thesis have been specifically chosen to enable feedback resource management. Software components are assumed to have a cyclic, primarily resource dependent, behavior, i.e., the component output is driven by the resources provided as input. This also means restricting software so that it cannot take resource decisions by itself, as this would make estimating the parameters in the input/output model difficult.

A recurring pattern has also been that resource management decisions are made small and assumed to be part of an infinite sequence of actions. This is typical for a feedback control approach, especially when including estimation, and this thesis demonstrates three approaches of how to break down resource management decisions in this manner.

Cross domain models

The thesis has demonstrated techniques for creating system models that encompass both physical and computational aspects of a system and how such models can be used in feedback-based resource management to control performance metrics that depend on multiple system components. Particular emphasis has been given to including energy and thermal properties, as these properties are a common concern, especially in mobile systems. The thermal model is validated through experiments on a mobile robot using adaptive resource management.

The discrete flow model used for resource control in Chapter 7 is applicable to both computational and physical resource flows. The event like nature of how computational resources are generated and consumed is handled through integration, meaning that their effect is well defined over a period of time.

Alternatives to hard real-time

The absolute guarantees required for hard real-time theory is not suited for the systems studied in this thesis. Neither are the systems mentioned of the type that are safety critical nor do they require perfect timeliness in order to function. As such, the focus is rather on describing how the performance varies with the allocated resources.

This thesis shows how to use rate-based performance metrics for both competitive and collaborative resource management and how event-based estimation techniques can be used to derive component model parameters online. The synchronization example in particular does not fit with the traditional hard/soft real-time formulations, as there are no deadlines involved in the metric at all.

Resource conservation and green computing

Reducing energy consumption in devices serves both to enable more advanced mobile systems and to build a more sustainable society. This thesis shows both how to explicitly relate power consumption to system metrics for a software system and how cooperative techniques can be employed to reduce the overall energy consumption in a population of co-located devices.

9.2 Future work

The field of cyber-physical systems is still new and much remains to be done in regards to the topics presented in this thesis.

Modeling of cyber-physical systems

The cyclic component model used in this thesis lacks the ability to describe more stateful components. Each cycle is assumed to take on average the same amount of resources, but as shown in the plot regarding rendering time for video frames, there are patterns that could be exploited for more exact prediction and control.

One possibility is to tie the cycle resource demand to a markov chain model, where each state represents a tighter estimate of the resource need. This would enable better prediction that would result in better performance for, e.g., synchronization control.

Competitive resource management

The components used in this chapter are simplistic and one natural extension is to used composite collaborative components, e.g., the conversational video pipeline from Chapter 7. The only constraint is that the utility function is concave, which is true, e.g., for pipeline throughput.

The potential uses for non-homogeneous utility functions is never explored in detail, though the algorithm does support it. Potential uses for this could be to treat critical components different than non-critical or to have different utility functions depending on the level of system overload.

The algorithm currently does not support multicore resource allocation, as this would introduce bin packing-like properties which would not be solvable by a gradient descent style algorithm. However, it would be interesting to see how a convex relaxation of a bin-packing problem could be used to efficiently do approximately optimal resource management on multicore platforms.

Collaborative resource management

In the example discussed in this thesis, the flows only go from physical (power) to computational (events) but never the other way, thereby never fully realizing the potential to model the end-to-end performance of an entire physical system. Such a system would be the result of having computational resources feed into physical components, e.g., the computational output from a software controller component to the analog input of a physical device it controls, and studying resource management from energy to controller performance would be an important extension.

Furthermore, the control strategies used in this thesis never uses knowledge about the stochastic parts of the model, e.g., the cycle resource requirements, even though they are known. Control performance could potentially be enhanced by stochastic control theory, though with special care to handle exponentially distributed disturbances rather than gaussian.

Cooperative resource management

The current cost function used in the heuristic algorithm could be generalized to prohibit specifically undesirable situations by introducing cross terms. The current

form

$$J = \sum_i (n_i - \bar{n})^2 + \sum_j (f_j - \bar{f})^2 \tag{9.1}$$

is rewritable as a quadratic form

$$J = \tilde{n}^T Q \tilde{n} + \tilde{n}^T R \tilde{f} + \tilde{f}^T S \tilde{f} \tag{9.2}$$

for some square matrixes Q, R, S, where $\tilde{n} = n - \bar{n}$ and $\tilde{f} = f - \bar{f}$. Particularly the cross terms, $(n_i - \bar{n})R_{i,j}(f_j - \bar{f})$ are interesting since these would include the cost for giving rare objects to clients who are almost done and therefore can leave the economy and conversely giving common objects to clients with few in possession.

The lack of a qualitative analytical model for throughput performance limits further analysis of the system behavior close to congestion. Such models could, for instance, be developed using Colored Petri Nets or similar types of transition systems, but one obstacle is the rapidly growing state space. The alternative route would be to describe the deficiencies of the arbitration algorithm as a throughput disturbance, trying to quantify this effect.

While the objects are divided in uniform pieces in this thesis, studying the market behavior in cases where objects are of different size is an interesting extension. Some care would have to be taken so to not give owners of small but rare objects unfair advantage when trading, specifically reducing the risk that the system would break down around who is forced to fetch larger objects from the remote repository.

E-logistics

The underlying economy of energy could be more explicitly used to form a currency-based economy. This would enable the trade of heterogeneous goods, e.g., trading a file object for a computation or a GPS measurement. This could potentially reduce long range communications even further, as clients without objects could make initial trades by offering computational services rather than fetching initial objects from the remote repository. One problem here would be to keep traders honest about costs and prevent monopoly situations, but given large enough populations this should be solvable.

Removing the need for a centralized decision mechanism is another important step for creating more flexible and resilient market places. One way to do this in an energy based economy is letting nodes solve the arbitration problem, allowing them to take part of the trades involved in the solutions they compute. Investigating how a client can benefit by creating energy savings for others seems like a very interesting research direction, one with many parallels to regular economics.

By extending the system with such an economy would make the entire scheme appear like a traditional logistics problem. Goods have production costs, from fetching them remotely, performing a calculations or using sensors, and transportation costs, i.e., the energy associated with communication. Clients could profit by acting

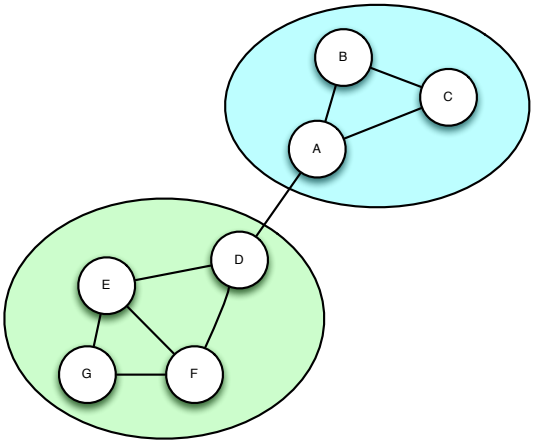


Figure 9.1 A more general problem structure where clients are divided into co-located subgroups, within each of which short range communication is possible, while only some clients would be close enough to communicate with neighboring groups.

as distributors, fetching goods from clients out of short range reach of the others (see Figure 9.1). This would require a more explicit model of the spatial relations between clients and the cost function to minimize would need reflect the varying total cost of services. There is likely traditional logistics theory that could apply to a system like that.

Bibliography

Abeni, L. and G. Buttazzo (1998a). "Integrating multimedia applications in hard real-time systems". In: *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*. Madrid, Spain, pp. 3–13.

Abeni, L. and G. C. Buttazzo (1998b). "Integrating multimedia applications in hard real-time systems". In: *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS '98)*. Madrid, Spain, pp. 4 –13.

Abeni, L. and G. C. Buttazzo (1999). "Adaptive bandwidth reservation for multimedia computing". In: *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*. IEEE Computer Society, Washington, DC, USA.

Abeni, L., G. Lipari, and G. C. Buttazzo (1999). "Constant bandwidth vs proportional share resource allocation". In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS '99)*. Vol. 2. Florence, Italy, pp. 107 –111.

Abeni, L., L. Palopoli, S. Superiore, and J. Walpole (2002). "Analysis of a reservation-based feedback scheduler". In: *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS 2002)*. Austin, Texas, USA, pp. 71–80.

Akyildiz, I. F., G. Morabito, and S. Palazzo (2001). "Tcp-peach: a new congestion control scheme for satellite ip networks". *IEEE/ACM Trans. Netw.* **9**:3, pp. 307–321. ISSN: 1063-6692.

Anderson, D. P., S. Tzou, R. Wahbe, R. Govindan, and M. Andrews (1990). "Support for continuous media in the DASH system". In: *Proceedings of the 10 International Conference on Distributed Computing Systems (ICDC '90)*. Berkeley, CA, USA, pp. 54 –61.

Android (2014). *Android.com*. URL: `http://www.android.com`.

Androutsellis-Theotokis, Stephanos, and D. Spinellis (2004). "A survey of peer-to-peer content distribution technologies". *ACM Computing Survey* **36**:4, pp. 335–371. ISSN: 0360-0300.

AQuoSA (2010). URL: `http://aquosa.sourceforge.net`.

Årzén, K.-E. (1999). "A simple event-based PID controller". In: *Proceedings of the 14th IFAC World Congress*.

Åström, K. J. and R. M. Murray (2008). *Feedback systems: an introduction for scientists and engineers*. Princeton University Press, 41 William Street, Princeton, New Jersey, p. 396.

Ausiello, G., A. D'Atri, and M. Protasi (1980). "Structure preserving reductions among convex optimization problems". *Journal of Computer and System Sciences* **21**:1, pp. 136–153.

Axelrod, R. (2006). *Evolution of cooperation*. Basic Books.

Aydin, H., R. Melhem, D. Mosse, and P. Mejia-Alvarez (2001). "Dynamic and aggressive scheduling techniques for power-aware real-time systems". In: *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pp. 95–105.

Aydin, H., R. Melhem, D. Mosse, and P. Mejia-Alvarez (2004). "Power-aware scheduling for periodic real-time tasks". *Computers, IEEE Transactions on* **53**:5, pp. 584–600. ISSN: 0018-9340.

Balas, E., S. Ceria, G. Cornuéjols, and G. Pataki (1996). "Polyhedral methods for the maximum clique problem". *Clique, Coloring and Satisfiability: The Second DIMACS Challenge. The American Mathematical Society, Providence, RI*, pp. 11–27.

Balasubramanian, N., A. Balasubramanian, and A. Venkataramani (2009). "Energy consumption in mobile phones: a measurement study and implications for network applications". In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. IMC '09. ACM, New York, NY, USA, pp. 280–293. ISBN: 978-1-60558-771-4.

Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield (2003). "Xen and the art of virtualization". *SIGOPS Oper. Syst. Rev.* **37**:5, pp. 164–177. ISSN: 0163-5980.

Bautista, D., J. Sahuquillo, H. Hassan, S. Petit, and J. Duato (2008). "A simple power-aware scheduling for multicore systems when running real-time applications". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–7.

Bhattacharyya, S. S., G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet (2009). "OpenDF: a dataflow toolset for reconfigurable hardware and multicore systems". *ACM SIGARCH Computer Architecture News* **36**:5, pp. 29–35.

Bini, E., G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Årzén, V. Romero Segovia, and C. Scordino (2011). "Resource management on multicore systems: the actors approach". *IEEE Micro* **31**:3, pp. 72–81.

Boyd, S. P. and L. Vandenberghe (2004). *Convex optimization*. Cambridge University Press, Cambridge, p. 716.

Branicky, M., S. Phillips, and W. Zhang (2002). "Scheduling and feedback co-design for networked control systems". In: *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*. Vol. 2, 1211–1217 vol.2.

Buttazzo, G. C. (1997). *Hard real-time computing systems: predictable scheduling algorithms and applications*. Kluwer Academic Publishers, Dordrecht, Netherlands.

Buttazzo, G. C., M. Spuri, and F. Sensini (1995). "Value vs. deadline scheduling in overload conditions". In: *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*. Pisa, Italy.

Caccamo, M., G. C. Buttazzo, and L. Sha (2000). "Elastic feedback control". In: *12th Euromicro Conference on Real-Time Systems (ECRTS 2000)*. Stockholm, Sweden, pp. 121–128.

Cervin, A. and T. Henningsson (2008). "Scheduling of event-triggered controllers on a shared network". In: *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pp. 3601–3606.

Cervin, A. and J. Eker (2003). "The control server: a computational model for real-time control tasks". In: *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*. Porto, Portugal, pp. 113 –120.

Cervin, A., J. Eker, B. Bernhardsson, and K. Årzén (2002). "Feedback–feedforward scheduling of control tasks". *Real-Time Systems* **23**, pp. 23–53.

Cha, H., J. Oh, and R. Ha (2003). "Dynamic frame dropping for bandwidth control in MPEG streaming system". *Multimedia Tools and Applications* **19**:2, pp. 155–178.

Chen, G., W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao (2008). "Energy-aware server provisioning and load dispatching for connection-intensive internet services." In: *NSDI*. Vol. 8, pp. 337–350.

Cheung, T. L., K. Okamoto, F. Maker, X. Liu, and V. Akella (2009). "Markov decision process (MDP) framework for optimizing software on mobile phones". In: *Proceedings of the Seventh ACM International Conference on Embedded Software*. ACM, New York, NY, USA, pp. 11–20.

Choudhury, P., P. Chakrabarti, and R. Kumar (2012). "Online scheduling of dynamic task graphs with communication and contention for multiprocessors". *Parallel and Distributed Systems, IEEE Transactions on* **23**:1, pp. 126–133. ISSN: 1045-9219.

Coffman Jr., E. G., M. R. Garey, and D. S. Johnson (1997). "Approximation algorithms for bin packing: a survey". In: *Approximation algorithms for NP-hard problems*. PWS Publishing Co, Boston, MA, USA, pp. 46 –93.

CGAL (2010). *Computational geometry algorithms library*. URL: `http://www.cgal.org`.

Cormen, T. H. (2001). *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, p. 1180.

Cucinotta, T., D. Giani, D. Faggioli, and F. Checconi (2011). "Providing performance guarantees to virtual machines using real-time scheduling". In: *Euro-Par 2010 Parallel Processing Workshops*. Springer, pp. 657–664.

D-Bus (2010). URL: `http://www.freedesktop.org/wiki/Software/dbus`.

Daniel E. Rivera, M. M. and S. Skogestad (1986). "Internal model control 4: pid controller design". In: *Industrial and Engineering Chemestry Research*, pp. 252–265.

Delvare, J. (2010). *Kernel driver lm83*. `http://www.mjmwired.net/kernel/Documentation/hwmon/lm83`.

Demers, A., S. Keshav, and S. Shenker (1989). "Analysis and simulation of a fair queueing algorithm". In: *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*. ACM, New York, NY, USA, pp. 1–12.

Eker, J. and J. W. Janneck (2003). *CAL language report*. Tech. rep. UCB/ERL M03/48. University of California at Berkeley.

Federal Communications Commission (2010). *MOBILE BROADBAND: THE BENEFITS OF ADDITIONAL SPECTRUM*. Tech. rep. Federal Communications Commission.

Ferreira, A. P., D. Mosse, and J. C. Oh (2007a). "Thermal faults modeling using a rc model with an application to web farms". In: *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS 2007)*. Pisa, Italy, pp. 113–124.

Ferreira, A., D. Mosse, and J. Oh (2007b). "Thermal faults modeling using a rc model with an application to web farms". In: *Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on*, pp. 113–124.

Foundry, D. (2010). *Tech analysis: crackdown 2 demo*. URL: `http://www.eurogamer.net/articles/digitalfoundry-crackdown2-demo-blog-entry`.

Fu, Y., N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang (2010a). "Feedback thermal control for real-time systems". In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. IEEE, pp. 111–120.

Fu, Y., N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang (2010b). "Feedback thermal control for real-time systems". In: *Proceedings of the 16th Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*. Stockholm, Sweden, pp. 111–120.

Geyer, T. (2005). *Low Complexity Model Predictive Control in Power Electronics and Power Systems*. English. Cuvillier Verlag, Göttingen, p. 291.

Giselsson, P., M. D. Doan, T. Keviczky, B. D. Schutter, and A. Rantzer (2013). "Accelerated gradient methods and dual decomposition in distributed model predictive control". *Automatica* **49**:3, pp. 829 –833. ISSN: 0005-1098.

GLPK. Gnu linear programming kit. URL: `http://www.gnu.org/software/glpk`.

Godsil, C. D., G. Royle, and C. Godsil (2001). *Algebraic graph theory*. Vol. 8. Springer New York.

Goel, A., J. Walpole, and M. Shor (2004). "Real-rate scheduling". In: *Proceedings of the 10 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*. Toronto, Canada, pp. 434 –441.

Golestani, S. J. (1994). "A self-clocked fair queueing scheme for broadband applications". In: *Proceedings of the 13th IEEE Conference on Networking for Global Communications (INFOCOM '94)*.

Grant, M. and S. P. Boyd (2010). *CVX: matlab software for disciplined convex programming, version 1.21*. `http://cvxr.com/cvx`.

Halperin, D., B. Greensteiny, A. Shethy, and D. Wetherally (2012). "Demystifying 802.11n power consumption". In: *Proceedings of the 2010 Workshop on Power Aware Computing and Systems*.

Harbour, M. G. (2008). *FRESCOR: framework for real-time embedded systems based on contracts*. URL: `http://www.frescor.org`.

He, T., J. A. Stankovic, M. Marley, C. Lu, Y. Lu, and T. Abdelzaher (2007). "Feedback control-based dynamic resource management in distributed real-time systems". *Journal of Systems and Software* **80**:7, pp. 997–1004.

Henningsson, T. and A. Cervin (2009). "Comparison of LTI and event-based control for a moving cart with quantized position measurements". In: *Proceedings of the European Control Conference*.

Herrtwich, R. G. (1991). "The role of performance, scheduling and resource reservation in multimedia systems". In: *Proceedings of the International Workshop on Operating Systems of the 90s and Beyond*. Springer-Verlag, London, UK, pp. 279–284. ISBN: 3-540-54987-0.

Hoffmann, H., J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal (2010). "Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments". In: *Proceedings of the 7th international conference on Autonomic computing*. ACM, pp. 79–88.

Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M (2010). Intel Corporation. Santa Clara, CA, USA.

Isovic, D. and G. Fohler (2004). "Quality aware MPEG-2 stream adaptation in resource constrained systems". In: *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004)*. Catania, Italy, pp. 23–32.

Jain, R., D.-M. Chiu, and W. R. Hawe (1984). *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Eastern Research Laboratory, Digital Equipment Corporation.

Jerez, J. L., K.-V. Ling, G. A. Constantinides, and E. C. Kerrigan (2012). "Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry". *Control Theory & Applications, IET* **6**:8, pp. 1029–1041.

Jin, X. and Y.-K. Kwok (2010). "Cloud assisted p2p media streaming for bandwidth constrained mobile subscribers". In: *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pp. 800–805.

Jung, E., Y. Wang, I. Prilepov, F. Maker, X. Liu, and V. Akella (2010). "User-profile-driven collaborative bandwidth sharing on mobile phones". In: *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. MCS '10. ACM, New York, NY, USA, 2:1–2:9. ISBN: 978-1-4503-0155-8.

Kaneko, H., J. A. Stankovic, S. Sen, and K. Ramamritham (1996). "Integrated scheduling of multimedia and hard real-time tasks". In: *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96)*. Washington, DC, USA.

Karp, R. (1972). "Reducibility among combinatorial problems". English. In: Miller, R. et al. (Eds.). *Complexity of Computer Computations*. The IBM Research Symposia Series. Springer US, pp. 85–103. ISBN: 978-1-4684-2003-6.

Katabi, D., M. Handley, and C. Rohrs (2002). "Congestion control for high bandwidth-delay product networks". In: *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '02. ACM, New York, NY, USA, pp. 89–102. ISBN: 1-58113-570-X.

Kellerer, H., U. Pferschy, and D. Pisinger (2004). *Knapsack problems*. Springer-Verlag Berlin, Heidelberg, Germany, p. 546.

Kreitz, G. and F. Niemela (2010). "Spotify – large scale, low latency, p2p music-on-demand streaming". In: *Proceedings of the 10th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–10.

Laszewski, G. von, L. Wang, A. Younge, and X. He (2009). "Power-aware scheduling of virtual machines in dvfs-enabled clusters". In: *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pp. 1–10.

Lee, E. A. (2006). "The problem with threads". *IEEE Computer* **39**:5, pp. 33 –42.

Lindberg, M. (2007). *A Survey of Reservation-Based Scheduling*. Tech. rep. ISRN LUTFD2/TFRT--7618--SE. Department of Automatic Control, Lund University, Sweden.

Lindberg, M. (2009). "Constrained online resource control using convex programming based allocation". In: *Proceedings of the 4th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2009)*.

Lindberg, M. (2010a). "A convex optimization-based approach to control of uncertain execution platforms". In: *Proceedings of the 49th IEEE Conference on Decision and Control*. Atlanta, Georgia, USA.

Lindberg, M. (2010b). "Convex programming-based resource management for uncertain execution platforms". In: *Proceedings of First International Workshop on Adaptive Resource Management*. Stockholm, Sweden.

Lindberg, M. (2013). "Feedback-based cooperative content distribution for mobile networks". In: *The 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. Barcelona, Spain.

Lindberg, M. (2014a). "A prototype implementation of an energy-conservative cooperative content distribution system". In: *The 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems.* In submission.

Lindberg, M. (2014b). "Analysis of a feedback-based energy conserving content distribution mechanism for mobile networks". In: *Proceedings of IFAC World Congress 2014*.

Lindberg, M. and K.-E. Årzén (2010a). "Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties". In: *Proc. 31st IEEE Real-Time Systems Symposium*. San Diego, CA.

Lindberg, M. and K. Årzén (2010b). "Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties". In: *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS 2010)*. San Diego, CA, USA.

Lipari, G. and S. K. Baruah (2001). "A hierarchical extension to the constant bandwidth server framework". In: *Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01)*. Taipei, Taiwan.

Maciejowski, J. M. (2002). *Predictive control: with constraints*. Pearson Education Limited, Edinburgh Gate, Harlow, p. 331.

Maggio, M., E. Bini, G. Chasparis, and K.-E. Årzén (2013). "A game-theoretic resource manager for rt applications". eng. In: Paris.

Makhorin, A. (2000). *Gnu linear programming kit*. `http://www.gnu.org/software/glpk`.

Marzario, C. P., T. Cucinotta, L. Palopoli, L. Marzario, and G. Lipari (2004). "Adaptive reservations in a Linux environment". In: *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 238 –245.

Matplotlib (2014). *Matplotlib*. URL: `http://matplotlib.org` (visited on 05/30/2013).

Mattingley, J. and S. Boyd (2012). "Cvxgen: a code generator for embedded convex optimization". English. *Optimization and Engineering* **13**:1, pp. 1–27. ISSN: 1389-4420.

McNamara, L., C. Mascolo, and L. Capra (2008). "Media sharing based on colocation prediction in urban transport". In: *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*. MobiCom '08. ACM, New York, NY, USA, pp. 58–69. ISBN: 978-1-60558-096-8.

Mercer, C. W., S. Savage, and H. Tokuda (1994). "Processor capacity reserves: operating system support for multimedia applications". In: *Proceedings of the International Conference on Multimedia Computing and Systems*, pp. 90 –99.

MobileRobots (2006). *Pioneer 3 Operations Manual*. MobileRobots Inc. Amherst, NH, US.

Model VSBC-8 Reference manual (2007). Versalogic Corporation. Eugene, OR, US.

Mok, A. K., X. Feng, and D. Chen (2001a). "Resource partition for real-time systems". In: *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*. Taipei, Taiwan, pp. 75–84.

Mok, A, X Feng, and D Chen (2001b). "Resource partition for real-time systems". In: *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTAS 2001)*. Taipei, Taiwan, pp. 75 –84.

Nagle, J. B. (1987). "On packet switches with infinite storage". *IEEE/ACM Transactions on Networking (ToN)* **35**:4, pp. 435 –438.

National Semiconductor Corporation (1999). *LM83 Triple-Diod Input and Logical Digital Temperature Sensor with Two-Wire Interface, DS101058*.

NetworkX (2013). *NetworkX*. URL: `http://networkx.github.io`.

OCERA (2010). URL: `http://www.ocera.org`.

OpenSSH (2014). *Openssh*. URL: `http://www.openssh.com`.

Parekh, A. K. and R. G. Gallager (2007). "A generalized processor sharing approach to flow control in integrated services networks: the single node case". In: *The Best of the Best : Fifty Years of Communications and Networking Research*. 1st ed. IEEE, pp. 533 –546.

Python (2014). *Python programming language*. URL: `http://python.org`.

Rajkumar, R., C. Lee, and D. Siewiorek (1997). "A resource allocation model for QoS management". In: *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*. San Francisco, CA, USA, pp. 298 –307.

Robert, Y. (2011). "Task graph scheduling". *Encyclopedia of Parallel Computing*, pp. 2013–2025.

Romero Segovia, V., K.-E. Årzén, S. Schorr, R. Guerra, G. Fohler, J. Eker, and H. Gustafsson (2010). "Adaptive resource management framework for mobile terminals-the actors approach". In: *Proceedings of the First International Workshop on Adaptive Resource Management (WARM), Stockholm, Sweden*.

Romero Segovia, V., M. Kralmark, M. Lindberg, and K.-E. Årzén (2011). "Processor thermal control using adaptive bandwidth resource management". In: *Proceedings of IFAC World Congress, Milan, Italy*. Milano, Italy.

Rostedt, S. and D. V. Hart (2007). "Internals of the rt patch". In: *Proceedings of the Linux symposium*. Vol. 2007. Citeseer.

Saewong, S. and R. Rajkumar (1999). "Cooperative scheduling of multiple resources". In: *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*. Phoenix, AZ, USA, p. 90.

Sandee, J., W. Heemels, and P. van den Bosch (2007). "Case studies in event-driven control". In: *Hybrid Systems: Computation and Control*. Vol. 4416. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 762–765.

Segovia, V. R. and K.-E. Årzén (2010). "Towards adaptive resource management of dataflow applications on multi-core platforms". In: *Proceedings Work-in-Progress Session of the 22nd Euromicro Conference on Real-Time Systems, ECRTS 2010*. Brussels, Belgium, pp. 13–16.

Sharma, V., A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu (2003). "Power-aware qos management in web servers". In: *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pp. 63–72.

Skadron, K., T. Abdelzaher, and M. Stan (2002). "Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management". In: *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pp. 17–28.

Skadron, K., M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan (2003). "Temperature-aware microarchitecture". In: *ACM SIGARCH Computer Architecture News*. Vol. 31. 2. ACM, pp. 2–13.

Spuri, M. and G. Buttazzo (1996). "Scheduling aperiodic tasks in dynamic priority systems". *Real-Time Systems* **10**:2, pp. 179–210.

Steinmetz, R. (1996). "Human perception of jitter and media synchronization". *Selected Areas in Communications* **14**:1, pp. 61 –72.

Stiliadis, D. and A. Varma (1998). "Latency-rate servers: a general model for analysis of traffic scheduling algorithms". *IEEE/ACM Transactions on Networking (ToN)* **6**:5, pp. 611–624.

Tabuada, P. (2007). "Event-triggered real-time scheduling of stabilizing control tasks". *Automatic Control, IEEE Transactions on* **52**:9, pp. 1680–1685. ISSN: 0018-9286.

Taymans, W., S. Baker, A. Wingo, R. S. Bultje, and S. Kost (2010). *Gstreamer application development manual*. URL: `http://gstreamer.freedesktop.org/data/doc/gstreamer/0.10.30/manual/manual.ps`.

Treiber, M., A. Hennecke, and D. Helbing (2000). "Congested traffic states in empirical observations and microscopic simulations". *Phys. Rev. E* **62** (2), pp. 1805–1824.

Wang, H., Z. Tian, and Q. Zhang (2010). "Self-tuning price-based congestion control supporting tcp networks". In: *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, pp. 1–6.

Wolfson, O., B. Xu, and R. M. Tanner (2007). "Mobile peer-to-peer data dissemination with resource constraints". In: *Proceedings of the International Conference on Mobile Data Management (2007)*.

Xenomai (2010). URL: `http://www.xenomai.org`.

Xingang, P, P Goyal, X Guo, and H Vin (1996). "A hierarchical CPU scheduler for multimedia operating systems". In: *Proceedings of the USENIX 2nd Symposium on OS Design and Implementation (OSDI '96)*, pp. 107–122.

Yaacoub, E., L. Al-Kanj, Z. Dawy, S. Sharafeddine, F. Filali, and A. Abu-Dayya (2012). "A utility minimization approach for energy-aware cooperative content distribution with fairness constraints". *Transactions on Emerging Telecommunications Technologies* **23**:4, pp. 378–392. ISSN: 2161-3915.

Zeilinger, M. N., C. N. Jones, D. M. Raimondo, and M. Morari (2009). "Real-time MPC–stability through robust MPC design". In: *Proceedings of the Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conferenc*.

Zhang, L. (1990). "Virtual clock: a new traffic control algorithm for packet switching networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 20. 4. ACM, pp. 19–29.

A

Listings

A.1 MIPC

```
#define MIPC_FAILED -1
#define MIPC_SUCCESS 1

#define MIPC_MODE_LOCAL 1
#define MIPC_MODE_INET 2

#define MIPC_MAX_MSG_SIZE 1024
#define MIPC_PORT_BASE 40000

#define MIPC_PATH "/tmp/mipc/"

struct mipc_msg {
        int size;
        void *data;
};

typedef struct mipc_msg mipc_message;
typedef int mipc_connection;

int mipc_create_socket(char *path);
int mipc_exists(char *path);
void* mipc_loop(void *arg);
int mipc_connect_server(int addr,
                        int (*recieve)(mipc_message *msg),
                        int mode);
mipc_connection mipc_connect_client(int addr);
mipc_connection mipc_connect_inet_client(char *hostname,
                                         int port);
int mipc_send(mipc_connection conn, mipc_message *msg);
```

```
mipc_message* mipc_new_message(void);
void mipc_free_message(mipc_message *msg);
```