Models and Complexity Results in Real-Time Scheduling Theory

PONTUS EKBERG
Abstract

When designing real-time systems, we want to prove that they will satisfy given timing constraints at run time. The main objective of real-time scheduling theory is to analyze properties of mathematical models that capture the temporal behaviors of such systems. These models typically consist of a collection of computational tasks, each of which generates an infinite sequence of task activations. In this thesis we study different classes of models and their corresponding analysis problems.

First, we consider models of mixed-criticality systems. The timing constraints of these systems state that all tasks must meet their deadlines for the run-time scenarios fulfilling certain assumptions, for example on execution times. For the other scenarios, only the most important tasks must meet their deadlines. We study both tasks with sporadic activation patterns and tasks with complicated activation patterns described by arbitrary directed graphs. We present sufficient schedulability tests, i.e., methods used to prove that a given collection of tasks will meet their timing constraints under a particular scheduling algorithm.

Second, we consider models where tasks can lock mutually exclusive resources and have activation patterns described by directed cycle graphs. We present an optimal scheduling algorithm and an exact schedulability test.

Third, we address a pair of longstanding open problems in real-time scheduling theory. These concern the computational complexity of deciding whether a collection of sporadic tasks are schedulable on a uniprocessor. We show that this decision problem is strongly coNP-complete in the general case. In the case where the asymptotic resource utilization of the tasks is bounded by a constant smaller than 1, we show that it is weakly coNP-complete.

Keywords: Real-time systems, Scheduling theory, Task models, Computational complexity

Pontus Ekberg, Department of Information Technology, Computer Systems, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.

© Pontus Ekberg 2015

ISSN 1651-6214
ISBN 978-91-554-9423-0
urn:nbn:se:uu:diva-267017 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-267017)
Till S. Ragnhild Elisabeth Ekberg,
d.v. och dä.
List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.


II  **Schedulability Analysis of a Graph-Based Task Model for Mixed-Criticality Systems.** Pontus Ekberg and Wang Yi. In *Real-Time Systems*, 2015, accepted for publication.


Reprints were made with permission from the publishers. All papers have been reformatted and minimally edited to fit the format of this thesis. I am the primary author and investigator of all the above papers.
Other publications


Acknowledgements

First I want to thank my Ph.D. advisor Wang Yi. He has given me an enormous and precious amount of freedom to pursue my interests, yet always been available for advice. Throughout all these years he has trusted me and supported me in all things and without exception. Thank you.

When I first started my Ph.D. studies, I was fortunate to enter a group where two other students, Martin Stigge and Nan Guan, were already working. Both of you are as friendly as you are brilliant. Having you as collaborators in my first years meant a lot to me and taught me much.

I can not recall a single time these last few years when going to work in the morning seemed much of a burden. This can largely be attributed to the welcoming atmosphere at the department in general and on the second floor in particular. I want to thank all of you that make lunch and “fika” such a pleasurable experience. The regulars not yet mentioned, past and present, include in no particular order: Pan Xiaoyue, Jonatan Lindén, Aleksandar Zeljić, Syed Md Jakaria Abdullah, Jonas Flodin, Peter Backeman, Philipp Rümmer, Kai Lampka, Pavel Krčál, Björn Forsberg, Morteza Mohaqeqi, Bengt Jonsson, Mingsong Lv, Pavle Subotic, Yi Zhang and Chuanwen Li.

I also want to thank all those who head the various parts of our department and our fantastic administrative staff. I am sure problems seem so few and far apart because you deal with them so efficiently.

Despite the strong support from inside the department, I wouldn’t have reached this point without those on the outside. Therefore I want to thank all of my family for their constant love and unfaltering support, before and during this time of my life. You are the best.

Last, but certainly not least, I want to thank my wife Sara and our daughter Selma. Sara, thank you for everything. Without you I would be lost. Selma, you are the strongest, kindest and bravest little girl in the world. If not for you I would likely have finished this thesis some time ago, but I would also be infinitely less happy. I love you.

This work was partially supported by UPMARC.
## Contents

1 Introduction ........................................................................................................... 11
   1.1 Modeling real-time systems ........................................................................ 12
   1.2 Proving properties of models ...................................................................... 13

2 Summary of contributions .................................................................................... 17
   2.1 Mixed-criticality systems .......................................................................... 17
   2.2 Resource-sharing systems .......................................................................... 21
   2.3 Complexity of foundational scheduling problems ..................................... 22

3 Conclusions and directions for future work ....................................................... 25

Summary in Swedish .................................................................................................. 27

References .................................................................................................................. 29

---

### PAPERS

**I  Bounding and Shaping the Demand of Generalized Mixed-Criticality Sporadic Task Systems** ................................................................. 33
   1 Introduction ....................................................................................................... 35
   2 Preliminaries .................................................................................................... 38
   3 Demand Bound Functions for Mixed-Criticality Tasks ............................... 41
   4 Tuning Relative Deadlines ............................................................................ 46
   5 Generalizing the Mixed-Criticality Task Model .......................................... 51
   6 Extending the Schedulability Analysis to the Generalized Task Model .......... 55
   7 Tuning Parameters for the Generalized Task Model ................................... 59
   8 Experimental Evaluation .............................................................................. 62
   9 Conclusions .................................................................................................... 74
   Appendix A  Proof of Lemma 4.1 ..................................................................... 77

**II  Schedulability Analysis of a Graph-Based Task Model for Mixed-Criticality Systems** ................................................................. 81
   1 Introduction ....................................................................................................... 83
   2 Model ............................................................................................................... 86
   3 Analysis .......................................................................................................... 91
   4 Tuning ............................................................................................................. 105
   5 Conclusions .................................................................................................... 107
1. Introduction

[...] the purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise.

– E. W. Dijkstra, *The Humble Programmer*

A common requirement put on computer systems is that they always should produce correct results. Another requirement put on many computer systems is that they should produce the expected results in a timely manner. For most of the systems we interact with directly, the latter requirement is not precisely defined and could roughly be stated as *fast enough to not be annoying*. How fast that is would depend on the application (and how easily you are annoyed!), perhaps a few seconds for loading a web page or $\frac{1}{50}$ of a second for computing the next frame in a video stream.

Most computer systems are, however, not in the form of what we might typically think of as “computers”. These are instead embedded in larger systems, such as vehicles, household appliances or medical and industrial equipment. For some of these embedded systems, timeliness of computation is a hard and precisely defined requirement. Take a car for example. Unless it is very old it will have embedded computer systems controlling various operations, such as the automatic breaking system and fuel injection. Computational tasks related to these operations must be guaranteed to complete within predefined time limits. Such computer systems are called real-time systems. The main challenge in designing real-time systems is not to achieve high speed per se, but rather to ensure highly predictable timing. Ideally, a real-time system’s conformance to given timing constraints should be proven with mathematical rigor.

This raises the question, how can one prove properties, in the mathematical sense, about, say, a car? In short, one can not. Cars are not mathematical objects and it is not possible to prove, in this sense, anything about them. What we can do is to prove properties about an abstract mathematical model of a car (or, in practice, a small subsystem of it).

The topic of this thesis is the study of such mathematical models and methods with which one can prove properties about them. The models we consider generally describe a set of computational tasks that together make up a real-time system. These models abstract away from all details that are not directly relevant for the timing properties. Notably, we do not care about the actual function of a computational task. Instead, we model only the temporal aspects,
such as how long it can take to perform its function, how often it needs to be
done and how soon it must be finished. In real-time scheduling theory, our
main concern is then how to order the various tasks that need to be performed
in a way that guarantees their conformance to the given timing constraints,
whenever this is possible.

1.1 Modeling real-time systems

In order to work on models of systems we must first define the formalism
in which those models can be described. A formalism generally consists of a
syntactic domain, which defines the ways models can be represented, and their
semantics, which imbue those representations with (abstract) meaning. In the
jargon of real-time scheduling theory, a task model refers to such a formalism.
For example, one of the most basic and commonly used task models is the
sporadic task model [23], which is informally defined below.

The sporadic task model

| Description: | In the sporadic task model, a system is modeled as a set
|              | of sporadic tasks, called a task system. Each task is an abstraction
|              | of a never-ending computational activity of the system, for example the
|              | periodic sampling of some sensor or the repeated execution of some
|              | planning algorithm. Each task is repeatedly activated at different time
|              | points. At those time points it is said to release a job, which is an
|              | abstraction of a single piece of computational workload. |
| Syntax:      | The representation of a sporadic task is an ordered triple of
|              | positive integers, \((e, d, p)\). These represent the worst-case execution
|              | time, the relative deadline and the period of the task, respectively. |
| Semantics:  | The integers \(e\) and \(p\) define the possible behaviors of the
|              | task in that each job can take up to \(e\) time units to finish and consecutive
|              | job releases by the task are separated by at least \(p\) time units. The
|              | integer \(d\) instead specifies the timing constraint: every job from the
|              | task has a deadline before which it must complete that is \(d\) time units
|              | after it’s release time. |

Each sporadic task defines a set of possible behaviors. Each behavior con-
sists of a sequence of jobs, potentially infinite in length, where each job is
characterized by a release time, a deadline and an execution time requirement.
It is common to illustrate sequences of jobs by drawing them along a time
line. Upwards-pointing arrows indicate job releases and downwards-pointing
arrows indicate the corresponding deadlines. The time intervals during which a job is executed in some schedule is marked with a box. As an example, Figure 1.1 shows a prefix of a possible behavior of a sporadic task system consisting of two tasks \((e_1, d_1, p_1)\) and \((e_2, d_2, p_2)\), where \((e_1, d_1, p_1) = (3, 4, 10)\) and \((e_2, d_2, p_2) = (4, 7, 8)\). The figure also shows a possible schedule, which satisfies the given timing constraints if the two tasks share a processor, i.e., if at most one job can execute at a time.

While the simple sporadic task model is perhaps the most commonly seen in real-time systems research, there exists a plethora of other task models. These generally offer greater flexibility and expressiveness than the sporadic task model, both in terms of the possible system behaviors they define and the timing constraints that should be satisfied.

1.2 Proving properties of models

The properties we want to prove about models of systems are usually in relation to some scheduling algorithm and some description of the computing platform. A scheduling algorithm is some well-defined procedure for deciding in which order to execute the waiting jobs. A computing platform defines the processing capacity, i.e., how jobs can be executed. For example, the platform could be a uniprocessor, where at most one job can execute at a time, or a multiprocessor, which can execute several jobs simultaneously. The computing platform could also specify features such as if jobs can be preempted or migrate between processors on a multiprocessor. Some of the questions that are commonly considered in the literature are listed below.
Schedulability: Given a task system $T$, a scheduling algorithm $A$ and a computing platform $C$, will all timing constraints of $T$ be met when scheduled by $A$ on $C$ for all possible behaviors of $T$?

Feasibility: Given a task system $T$ and a computing platform $C$, does there exist a scheduling algorithm $A$, such that $T$ is schedulable on $C$ with $A$?

Optimality: Given a scheduling algorithm $A$, a computing platform $C$ and a task model, are all task systems that can be expressed with the given task model schedulable with $A$ on $C$ if they are feasible on $C$?

How, then, can questions such as the above be answered? If we consider again the simple sporadic task model, it is easy to see that brute-force approaches are generally insufficient. Even a single sporadic task gives rise to an infinite number of behaviors, each of which can stretch on infinitely long in time. In fact, a single sporadic task can generate an uncountable set of distinct sequences of jobs. The development of abstractions and methods for dealing with challenges like this lies at the core of real-time scheduling theory. An example of a well-known result due to Dertouzos [15] is that the scheduling algorithm called Earliest Deadline First (EDF) is optimal on preemptive uniprocessors for a fairly large range of task models, including the sporadic task model. EDF is very straightforward. The scheduling rule is simply that, when given a choice between several jobs waiting to be executed, always execute the waiting job with the earliest deadline.

Another classic result due to Liu and Layland [22] is as follows. First, a sporadic task system $T$ is said to have implicit deadlines if $d = p$ for all $(e, d, p) \in T$, constrained deadlines if $d \leq p$ for all $(e, d, p) \in T$ and arbitrary deadlines otherwise. Then, a sporadic task system $T$ with implicit deadlines is feasible on a preemptive uniprocessor if and only if

$$U(T) \leq 1,$$

where

$$U(T) \overset{\text{def}}{=} \sum_{(e, d, p) \in T} \frac{e}{p}$$

is the utilization of $T$.

With the above we have an elegant and easy method for proving whether a given sporadic task system with implicit deadlines can be scheduled on a preemptive uniprocessor. If the formula in Equation (1.1) is true, then yes, it can be successfully scheduled by EDF. Such a method is usually called a test.

---

1Liu and Layland’s result was actually for the related periodic task model. Baruah et al. [9] later showed that it also holds for the sporadic task model.
However, even small changes to the assumptions on the task model or computing platform can give rise to striking differences in the analysis complexity. For instance, another well-established result due to Baruah et al. [9] is that a sporadic task system $T$ (with deadlines that may be constrained or arbitrary) is feasible on a preemptive uniprocessor if and only if the formula in Equation (1.1) is true and additionally

$$\forall \ell \in \{0, 1, \ldots, \mathcal{P}(T) + \max\{d \mid (e, d, p) \in T\}\}, \quad \text{dbf}(T, \ell) \leq \ell, \quad (1.2)$$

where

$$\text{dbf}(T, \ell) \overset{\text{def}}{=} \sum_{(e, d, p) \in T} \max\left\{0, \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right\} \cdot e$$

is the demand bound function of $T$ in time interval lengths $\ell$, and

$$\mathcal{P}(T) \overset{\text{def}}{=} \text{lcm}\{p \mid (e, d, p) \in T\}$$

is $T$’s hyper-period.

Notice that while this latter result certainly is elegant as well, it has become considerably more complicated, even though we only dropped the assumption of implicit deadlines. We will revisit the computational complexity of evaluating the formula in Equation (1.2), as that is the topic of Papers IV and V in this thesis.

Generally, the more expressive the task model, the harder it becomes to prove properties about it. Ideally, questions such as those about schedulability and feasibility should be answered exactly. For example, a test for determining schedulability should answer “yes” if and only if all timing constraints are guaranteed to be met and “no” otherwise. Sometimes, however, it is not possible to answer those questions exactly. Perhaps no method is yet known for the given combination of task model, computing platform and scheduling algorithm, or the problem simply has a too high computational complexity to be practically answered except for small instances. In those situations an approximate answer is required, but the approximation must ensure the safety of the answers. A sufficient, but not exact, method for proving schedulability could answer “yes” if it can be determined that all timing constraints will be met and “maybe” otherwise. With such a test we face the risk of rejecting some systems that are actually schedulable, but we never risk mistaking an unschedulable system for a schedulable one.
2. Summary of contributions

The contributions of this thesis can be classified into three main areas. For each area, a short background and a summary of the contributions are given below. Papers I and II concern mixed-criticality systems; these are summarized in Section 2.1. The topic of Paper III is resource-sharing systems; it is summarized in Section 2.2. Papers IV and V settle two longstanding questions about the computational complexity of the uniprocessor feasibility problem for sporadic task systems; they are summarized in Section 2.3.

2.1 Mixed-criticality systems

Loosely speaking, a mixed-criticality system is a system where several functionalities of different importance, or criticality, co-exist on the same platform. For example, an airplane can contain, at the same time, computerized functionalities that are important for the safe operation of the airplane as well as others that are “merely” important for, say, fuel efficiency.

In the context of real-time scheduling theory, this would translate into a task system consisting of co-existing tasks that are not all equally important for ensuring safe or correct operation of the system. There are several ways in which this high-level idea of a mixed-criticality system can be formalized, and there have been discussions in the research community about which views are most practically useful (see, e.g., [12, 19, 11]). The dominating formalization in the literature is the one introduced by Vestal [28]. That is the one we have used and generalized in Papers I and II.

The main motivation of Vestal’s task model is the difficulty of safely estimating the worst-case execution time of a program on a given computer platform, e.g., to find the parameter $e$ of a sporadic task $(e, d, p)$. Worst-case execution time analysis is an active area of research in its own right [29]. Somewhat simplified, the estimation of the worst-case execution time of a program can be carried out in two ways. One is to formally analyze the program with a model of the platform, often using static analysis techniques. The other is to extensively test the program by executing it on the actual platform and to record the longest observed execution time. With formal analysis it is in many cases possible to deduce a value that is provably an upper bound on the execution time of the program, but depending on the complexity of the program and the platform that value can easily be several times higher than the true worst-case execution time. With measurement-based techniques the problem
is instead that no guarantee can be made about having observed the worst-case execution time during testing, and the value produced might therefore be an underestimation. Good engineering practice is to inflate the measured value, perhaps by 20%, as an added safety margin, but this in no way guarantees that the value is not an underestimation.

How these values are produced is not in itself a concern of scheduling theory, but Vestal’s observation was that system designers may prefer one way or another for different functionalities. For example, a system designer may be perfectly happy with the level of assurance provided by measurement-based approaches for most tasks, but may want to use formally proven upper bounds for safety-critical tasks. The question is then how to efficiently schedule tasks of different criticalities together, such that each one is guaranteed to meet its deadlines under the level of assurance requested for that functionality. In Vestal’s mixed-criticality sporadic task model, this is addressed by assigning each task a criticality level and giving it several worst-case execution time estimates, up to one per criticality level in the system. More formally, a mixed-criticality sporadic task in a system with \( k \) criticality levels is given by a tuple \((e(1), \ldots, e(k)), d, p, L\), where

- \( e(i) \) is its worst-case execution time estimate at the level of assurance mandated at criticality level \( i \),
- \( d \) and \( p \) are the relative deadline and period, respectively, and
- \( L \) is the task’s own criticality level.

As higher levels of assurance require stricter analysis of execution times, it is assumed that \( e(i) \geq e(j) \) for \( i > j \). A given run of a mixed-criticality sporadic task system \( T \) is said to be of criticality \( i \) when \( i \) is the smallest number such that for all tasks \((e(1), \ldots, e(k)), d, p, L\) \( \in T \), the actual execution time of each job of that task is no larger than \( e(i) \). Further, a run of criticality \( i \) is said to be successfully scheduled if and only if each task \((e(1), \ldots, e(k)), d, p, L\) \( \in T \) with \( L \geq i \) meets the deadlines of all its jobs.

The main challenge in the scheduling of such task systems is that information about the actual execution times of jobs is not gained until they have been executed. Therefore, there exists an inherent trade-off between first executing the most urgent jobs as opposed to the jobs that will provide the most useful information. However, it is typically assumed that the execution times can be monitored, so that it is possible, for example, for the scheduling algorithm to react to a job having executed for \( e(i) \) time units and not having finished yet.

An equivalent and useful way of thinking about mixed-criticality systems is that we are given a set of assumptions \( \{A_1, \ldots, A_k\} \) and a set of timing
constraints \{C_1, \ldots, C_k\}, where

\[ A_i \stackrel{\text{def}}{=} \left\{ \begin{array}{l}
\text{For all tasks } ((e(1), \ldots, e(k)), d, p, L) \in T, \\
\text{its jobs take no longer than } e(i) \text{ time units to finish.}
\end{array} \right. \]

\[ C_i \stackrel{\text{def}}{=} \left\{ \begin{array}{l}
\text{For all tasks } ((e(1), \ldots, e(k)), d, p, L) \in T \text{ with } L \geq i, \\
\text{its jobs meet their deadlines.}
\end{array} \right. \]

The scheduling problem is then to schedule the jobs of the system such that the implication \( A_i \to C_i \) holds for all \( i \in \{1, \ldots, k\} \), without any prior knowledge about which assumptions actually hold for the current run of the system. Knowledge about which assumptions hold can only be gained by executing jobs and observing their execution times.

In the literature, this view of mixed-criticality systems is often interpreted in terms of operational modes of the system (or rather, the scheduler). The system’s start mode is characterized by the lack of knowledge about the truth of the assumptions \{A_1, \ldots, A_k\}. While in the start mode, any scheduler must therefore fulfill all timing constraints \{C_1, \ldots, C_k\} or risk the falsity of some implication \( A_i \to C_i \). However, as soon as any job of some task \((e(1), \ldots, e(k)), d, p, L\) is observed to have executed for \( e(i) \) time units without having finished yet, it is clear that \( A_i \) does not hold for this run of the system and therefore that \( C_i \) does not need to be fulfilled. At that point the system (or scheduler) would enter a new operational mode in which all tasks with criticality level less than or equal to \( i \) can safely be ignored.

It has turned out that scheduling mixed-criticality sporadic task systems is very challenging. A fairly large number of scheduling algorithms have been proposed for such systems (see, e.g., [21, 5, 7] for a small sample or [13] for a survey), but none has been proven to be optimal for any of the common types of computing platforms. Also, almost all schedulability tests proposed for those scheduling algorithms are only sufficient. As a consequence of the apparent hardness of these problems, the focus in the literature has been on presenting scheduling algorithms with sufficient schedulability tests that are reasonably efficient and reasonably precise. Usually, an empirical evaluation is used to measure the precision of a sufficient schedulability test, but it may also be possible to prove that such a test has bounded deviation from some optimal behavior.

**Contributions**

In Paper I we study a scheduling algorithm for scheduling mixed-criticality systems on preemptive unprocessors. This algorithm is essentially the same as the algorithm \textit{EDF with virtual deadlines} (EDF-VD), which was first described by Baruah et al. [6], but it uses a more flexible method for assigning virtual deadlines to tasks. The idea behind EDF-VD is to view a mixed-criticality system in terms of operational modes as described above, and to
assign different (virtual) relative deadlines to tasks depending on the current mode. The virtual deadlines of a task are never larger than the task’s real relative deadline. Therefore, if the virtual deadlines are met, the real deadlines are also met. By assigning a smaller virtual deadline to a task of a high criticality level, the jobs of that task are prioritized higher and will possibly execute earlier than under plain EDF. Executing jobs of higher criticality earlier is desirable to some degree as that means information about their execution times is gained earlier and there is more time to react to any assumptions shown to be false.

We adapt the concept of demand bound functions [9] to this setting by defining them on a per-mode basis. The main difficulty in doing so is that modes are not independent because workload can be carried over from one mode to the next. Calculating the exact influence of one mode to the next is very demanding and therefore our formulation of demand bound functions includes an over-approximation of the workload that can be carried over between modes. In this way we can build a sufficient schedulability test that, given an assignment of virtual deadlines to tasks, scales similarly to schedulability tests for normal sporadic tasks under EDF. In other words, it runs in pseudo-polynomial time if the utilization of the task system in each criticality mode is bounded by some constant $c$, such that $c < 1$.

Our formulation of demand bound functions has the interesting property that the virtual deadlines act as tuning knobs with which we can shift demand from one mode to the next. By decreasing the virtual deadline used by a task in one mode, we increase the demand bound function of that task in that mode, but at the same time decrease its demand bound function in the next mode. The problem of finding virtual deadlines that ensure the schedulability of the task system can then be thought of as the problem of turning those knobs until the sum of the demand bound functions in each mode fall below the supply bound function of the computing platform. In Paper I we also present an efficient heuristic technique for doing so.

This approach is then empirically evaluated in Paper I by comparing the resulting schedulability test with the schedulability tests of other algorithms proposed in the literature. This is done by randomly generating large numbers of synthetic task systems and trying all schedulability tests on them. We found that our schedulability test often admitted significantly more task systems than previous tests.

Lastly, in Paper I we generalize Vestal’s mixed-criticality task model by letting all task parameters, not just the worst-case execution time, be subject to uncertainties. In the generalized task model we also make it possible for new tasks to be activated in each operational mode and allow the operational modes themselves to be structured as any directed acyclic graph, instead of having a linear structure. We show how our scheduling and analysis approach can be adjusted to work for this more general task model.
In Paper II we present an even more general task model, which we obtain by merging generalized mixed-criticality semantics with the digraph real-time task model (DRT) [26]. The DRT task model is a generalization of, among others, the sporadic task model. In DRT, each task is represented by an arbitrary directed graph, with vertices representing job types (with worst-case execution time and relative deadline) and edges representing the possible orders in which different jobs of the task can be released. The resulting task model, which we call mode-switching DRT (MS-DRT), is very versatile. It can be used to model complicated tasks and a large number of different types of mode switches, and also systems that combine criticality levels with some more classic operational modes. In MS-DRT there are no limitations on the order in which different modes can be entered. Cyclic mode changes are also allowed.

The challenge with such a general task model is to analyze the resulting system models. We show that by carefully introducing abstractions that over-approximate the interactions between modes it is possible to apply existing analysis methods for DRT to achieve a sufficient and reasonably scalable schedulability test for MS-DRT with EDF.

2.2 Resource-sharing systems

In basic task models, the jobs released by different tasks are completely independent of each other. This means that a scheduler is allowed to preempt an already executing job at any time to run another job and later resume the preempted job. In many real systems, however, such behavior is not always allowed. For example, different programs may share a common data structure that needs to be kept in a consistent state and therefore is protected by a mutex or similar. Other programs may use some non-preemptable hardware device and disallow preemptions during certain parts of their execution. In real-time scheduling theory, such systems are called resource-sharing systems. The name may sound odd because all scheduling concern systems that share resources in one way or another; the “resource” in resource-sharing systems refer in particular to those resources that are shared non-preemptively (or are locked), like a mutex-protected data structure.

The sporadic task model, and others, can be extended to model resource-sharing systems. A system model could then consists of a set of tasks, a set of resources and a mapping specifying which tasks may use which resources. The mapping would also specify for how long a job from a given task may lock a given resource. Jobs are typically allowed to lock several resources at a time.

There have been several results published about resource sharing with simple task models, such as the sporadic task model. For example, the classic priority inheritance protocol (PIP) and priority ceiling protocol (PCP) provide
extra scheduling rules for the fixed-priority scheduling algorithm to work well with resource-sharing systems [25]. For dynamic scheduling algorithms, the stack resource policy (SRP) plays a similar role [1]. The combination of EDF and SRP is in fact an optimal (online) scheduling algorithm for sporadic task systems with shared resources on otherwise preemptive uniprocessors [2].

For more complicated task models, resource-sharing is not as well understood. Scheduling algorithms have been proposed for DRT task systems with shared resources, but these are suboptimal and have only sufficient schedulability tests [20]. Resource sharing has also been considered for expressive task models based on timed automata [18]. These are analyzed using model checking techniques, which can provide exact answers, but generally do not scale well.

**Contributions**

In Paper III we consider the GMF-R task model, which is the generalized multiframe (GMF) task model extended with resource sharing. The plain GMF task model is a generalization of the sporadic task model and also a special case of the DRT task model [8]. The GMF task model is equivalent to the DRT task model restricted to tasks represented by directed cycle graphs. In GMF-R, every job type (vertex) has its own mapping to describe the resources it can use, so different jobs of the same task can access different resources.

The main contributions of Paper III are two-fold. First, we propose the resource deadline protocol (RDP), which we combine with EDF to form the EDF+RDP scheduling algorithm. We prove that EDF+RDP is optimal for the GMF-R task model on preemptive uniprocessors. Second, we describe a feasibility test for GMF-R on preemptive uniprocessors and prove that the test is exact. The feasibility test has similar scalability as previous tests for plain GMF, i.e., it runs in pseudo-polynomial time if the utilization of the task system is upper-bounded by some constant \( c \), such that \( c < 1 \). In addition, we prove that EDF+RDP enables stack sharing, like EDF+SRP, and that it limits the number of preemptions to at most one per job release.

**2.3 Complexity of foundational scheduling problems**

Classifying the computational complexity of problems offer valuable insights into why certain problems are hard and how to best tackle them. Many problems in real-time scheduling theory are known to be hard in the sense that they have been proven to belong to some complexity class that is generally believed to be hard. Still, some of the most basic scheduling problems have eluded such classification. One of these is the problem of deciding whether a given sporadic task system is feasible on a preemptive uniprocessor. This problem is
relevant on its own, and also appears as a special case of many other problems in real-time scheduling theory.

We have seen in Section 1.2 that the feasibility problem for sporadic task systems with implicit deadlines can be solved in polynomial time by simply evaluating the formula in Equation (1.1). We have also seen that when deadlines are not implicit, the feasibility problem can be solved by evaluating the formula in Equation (1.2) instead. Equation (1.2) immediately yields an exponential-time feasibility test and also shows that the feasibility problem is in $\text{coNP}$. Additionally, it is known that if the utilization of the task set is bounded from above by some constant $c$, such that $c < 1$, the feasibility problem can be solved in pseudo-polynomial time [9]. These results have been known since 1990 and since then they have been widely cited and used in the literature, but pinpointing the computational complexity of the sporadic feasibility problem remained an open problem. In 2010, Baruah and Pruhs listed it as one of the “most important open algorithmic problems in real-time scheduling” [10]. Shortly thereafter, Eisenbrand and Rothvoß [16] provided the first lower bound on the complexity of this problem by proving weak $\text{coNP}$-hardness of the general case with unbounded utilization. This still left questions unanswered, mainly the following two.

(i) Does a pseudo-polynomial time algorithm exist for the general case?

(ii) Does a polynomial time algorithm exist for the restricted case where the utilization is bounded by a constant?

Eisenbrand and Rothvoß conjectured positive answers for both of the above questions [16].

**Contributions**

We have shown that the answers to both of the above questions are negative unless $P = NP$.

In Paper IV we address the general case of the feasibility problem for sporadic task systems on preemptive unprocessors, and show that it is strongly $\text{coNP}$-complete. This holds even if restricted to task systems $T$ where all tasks $(e, d, p) \in T$ have $d \leq p$ and $e = 1$. From these results it follows that no pseudo-polynomial time algorithm exists for this problem unless $P = NP$.

In Paper V we instead address the restricted case in which every task system $T$ has $U(T) \leq c$, for some constant $c$. We show that this problem is weakly $\text{coNP}$-complete for any constant $c$, such that $0 < c < 1$. This holds even if all tasks $(e, d, p) \in T$ have $d \leq p$. Unless $P = NP$, it follows that no polynomial time algorithm exists for this problem.
3. Conclusions and directions for future work

There is little doubt that mixed-criticality systems are becoming increasingly common. From a practical point of view, many challenges remain in finding the right abstractions and bridging the gaps between the different parts that make up the design of such systems, of which scheduling is just one. From a theoretical point of view, many results have already been produced, like those in this thesis, but most fundamental questions about Vestal’s task model, and similar ones, remain unanswered [17]. Some examples are below.

(i) Which scheduling algorithms are optimal for mixed-criticality tasks, and in which settings?
(ii) Are there optimal scheduling algorithms with low overhead costs?
(iii) How does one exactly decide feasibility or schedulability for existing scheduling algorithms, and what is the computational complexity of those problems?

Problem (ii) can be understood in terms of practical overhead costs (e.g., the number of preemptions), but also in terms of the run-time complexity of the scheduling algorithm itself. We formalize the latter notion by defining it in terms of the cumulative run-time of all activations of the scheduling algorithm in some time interval.

Definition. We say that \( f(\ell, n) \) is the scheduling complexity of a scheduling algorithm \( A \) if \( f(\ell, n) \) is the maximum cumulative running time of \( A \)’s scheduling decisions for any task system \( T \) in any time interval \([t_1, t_2)\), such that \( \ell = t_2 - t_1 \geq 1 \) and \( n \) is the size of the representation of \( T \).

For example, the scheduling complexity of EDF for feasible sporadic task systems with constrained deadlines is clearly in \( O(\ell n \log(n)) \).\(^1\) To the best of our knowledge, nothing similar to scheduling complexity has been extensively studied in the literature before and we believe that it is an interesting general direction for future work. For now, we make the following conjecture related to (ii), based on results of Baruah et al. [4] about the hardness of deciding feasibility of a finite collection of mixed-criticality jobs.

Conjecture. No optimal (online) scheduling algorithm for mixed-criticality sporadic task systems on preemptive uniprocessors has scheduling complexity in \( O(q(\ell, n)) \) for some two-variable polynomial \( q \), unless \( P = NP \).

\(^1\)A tighter bound might be possible. For periodic task systems or sporadic task systems restricted to integer release times, the scheduling complexity is in \( O(\ell \log(n)) \).
Many questions remain about the theories that arise around other task models or computing platforms as well. There seems to be a near endless supply of task models to consider, as new ones are often proposed by researchers (see, e.g., [14, 24, 27, 3] for a small sample of task models proposed in the recent years). Still, it is particularly important to answer fundamental questions about the most basic models upon which much of our understanding is based. A couple of those questions have been answered in this thesis, but some remain open. Below are two that are related to the work in this thesis.

(a) What is the computational complexity of the fixed-priority schedulability problem for sporadic task systems on preemptive uniprocessors, when deadlines are implicit or constrained?

(b) What is the computational complexity of the partitioned preemptive multiprocessor feasibility problem for sporadic task systems?

Let the decision problem in (b) be denoted PF. It is easy to reduce BIN-Packing to PF and therefore PF is NP-hard. If restricted to task systems with implicit deadlines, PF is in fact NP-complete. However, for constrained or arbitrary deadlines it is clear that PF is also coNP-hard because the corresponding uniprocessor feasibility problem can be reduced to it. Unless NP = coNP, it follows that PF can not be NP- or coNP-complete. However, we can find an upper bound because trivially PF ∈ \( \Sigma^p_2 \). We make the following conjecture.

**Conjecture.** PF is \( \Sigma^p_2 \)-complete already for two processors. □
Idag finns datorsystem inbyggda i allt från tvättmaskiner till bilar och robotar. Alla datorsystem bör fungera korrekt och när de ingår i ett säkerhetskritiskt system är det önskvärt att kunna bevisa detta med matematisk precision. För många inbyggda system betyder “korrekt” inte bara att rätt svar beräknas, utan också att rätt svar beräknas inom givna tidsramar. Sådana system brukar kallas realtidssystem.

För att kunna bevisa något om realtidssystem behöver vi först skapa matematiska modeller av dem. Inom schemaläggningssteori för realtidssystem studeras olika sätt att modellera realtidssystem samt metoder för att bevisa egenskaper hos de resulterande modellerna, särskilt med tanke på tidsaspekten. Oftast abstraheras allt som inte har med tid att göra bort från modellerna. De talar istället bara om saker som hur ofta en viss beräkning eller operation måste genomföras, hur lång tid det kan ta och hur snabbt det måste bli klart. Mycket fokus ligger sedan på i vilken ordning de olika processerna som utgör det modellerade realtidssystemet ska utföras, det vill säga hur de ska schemaläggas, för att alla tidsramar ska hållas.

Ju mer uttryckskraft som finns i modelleringsspråken, desto komplexare modeller kan skapas och desto svårare blir det att analysera dem. En utökning av befintliga modelleringsspråk som fått mycket uppmärksamhet under de senaste åren är att samtidigt beakta processer med olika kritikalitetsnivåer inom ett enda realtidssystem. Processer med olika kritikalitetsnivåer är inte lika viktiga för systemet, exempelvis kan det vara så att enbart ett fåtal av dem är säkerhetskritiska. För processer med hög kritikalitetsnivå måste tidsramarna hållas även under mycket pessimistiska antaganden om exempelvis exekveringstid, medan mindre pessimistiska (och kanske mer realistiska) antaganden är tillräckliga för processer med lägre kritikalitetsnivå. En stor utmaning med sådana system är att bevisa att samtliga tidskrav, vilka är baserade och villkorsliga på olika antaganden, är samtidigt uppfyllda, utan att i förväg veta vilka antaganden som stämmer.
Inom den här avhandlingen studeras bland annat nya metoder, algoritmer och uttrycksfulla modelleringsspråk för schemaläggning. En del av dessa är för system med olika kritikalitetsnivåer (uppsats I och II), andra är för system med processer som tidvis kan låsa resurser, exempelvis delade datastrukturer, exklusivt till sig själva (uppsats III).

References


Acta Universitatis Upsaliensis

_Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology_ 1324

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology”.)