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.y. 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
III  An Optimal Resource Sharing Protocol for Generalized Multiframe Tasks ........................................ 123
  1  Introduction ......................................................... 125
  2  Preliminaries ....................................................... 126
  3  The Resource Deadline Protocol ................................. 130
  4  Properties of EDF+RDP ............................................ 133
  5  Feasibility and Optimality ........................................ 135
  6  Complexity of the Test ............................................ 141
  7  Implementing EDF+RDP ............................................ 144
  8  Related Work ...................................................... 145
  9  Conclusions ........................................................ 146

IV  Uniprocessor Feasibility of Sporadic Tasks with Constrained Deadlines is Strongly coNP-complete ................ 149
  1  Introduction ........................................................ 151
  2  Preliminaries ........................................................ 153
  3  The Hardness of Sporadic Feasibility ............................ 154
  4  Conclusions ........................................................ 163

V  Uniprocessor Feasibility of Sporadic Tasks Remains coNP-complete Under Bounded Utilization .................... 165
  1  Introduction ........................................................ 167
  2  Preliminaries ........................................................ 170
  3  Reducing FEASIBILITY to c-FEASIBILITY ....................... 171
  4  Conclusions ........................................................ 187
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.

<table>
<thead>
<tr>
<th>The sporadic task model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> 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.</td>
</tr>
<tr>
<td><strong>Syntax:</strong> 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.</td>
</tr>
<tr>
<td><strong>Semantics:</strong> 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.</td>
</tr>
</tbody>
</table>

Each sporadic task defines a set of possible behaviors. Each behavior consists 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.

![Figure 1.1. A prefix of one possible behavior of the tasks \((e_1, d_1, p_1)\) and \((e_2, d_2, p_2)\).](image)

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

$$U(T) \leq 1,$$  \hspace{1cm} (1.1)

where

$$U(T) \equiv \sum_{(e,d,p) \in T} \frac{e}{p}$$

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

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

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 problem instances of small size. 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

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

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

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 uniprocessors. This algorithm is essentially the same as the algorithm *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 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 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 uniprocessors, and show that it is strongly 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 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 $O(\ell \log(n))$.\(^1\) 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 $P = NP$.\(^2\)

\(^1\)A tighter bound might be possible. For periodic task systems or sporadic task systems restricted to integer release times, the scheduling complexity is in $O(\ell \log(n))$.\(^2\)
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 ∈ Σ^P_2. We make the following conjecture.

**Conjecture.** PF is Σ^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 realltidssystem.

För att kunna bevisa något om realltidssystem behöver vi först skapa matematiska modeller av dem. Inom schemaläggningssteori för realltidssystem studeras olika sätt att modellera realltidssystem 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 realltidssystemet 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 realltidssystem. 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 exekveeringstid, 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 villkoriga på olika antaganden, är samtidigt uppfyllda, utan att i förväg veta vilka antaganden som stämma.
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”.)
Bounding and Shaping the Demand of Generalized Mixed-Criticality Sporadic Task Systems

Pontus Ekberg and Wang Yi

Abstract

We generalize the commonly used mixed-criticality sporadic task model to let all task parameters (execution-time, deadline and period) change between criticality modes. In addition, new tasks may be added in higher criticality modes and the modes may be arranged using any directed acyclic graph, where the nodes represent the different criticality modes and the edges the possible mode switches. We formulate demand bound functions for mixed-criticality sporadic tasks and use these to determine EDF-schedulability. Tasks have different demand bound functions for each criticality mode. We show how to shift execution demand between different criticality modes by tuning the relative deadlines. This allows us to shape the demand characteristics of each task. We propose efficient algorithms for tuning all relative deadlines of a task set in order to shape the total demand to the available supply of the computing platform. Experiments indicate that this approach is successful in practice. This new approach has the added benefit of supporting hierarchical scheduling frameworks.

1 Introduction

An increasing trend in real-time systems is to integrate functionalities of different criticality, or importance, on the same platform. Such mixed-criticality systems lead to new research challenges, not least from the scheduling point of view. The major challenge is to simultaneously guarantee temporal correctness at all different levels of assurance that are mandated by the different criticalities. Typically, at a high level of assurance, we need to guarantee correctness under very pessimistic assumptions (e.g., worst-case execution times from static analysis), but only for the most critical functionalities. At a lower level of assurance, we want to guarantee the temporal correctness of all functionalities, but under less pessimistic assumptions (e.g., measured worst-case execution times).

We adapt the concept of demand bound functions (Baruah et al., 1990) to the mixed-criticality setting, and derive such functions for mixed-criticality
sporadic tasks. These functions can be used to establish whether a task set is schedulable by EDF on a uniprocessor. In the mixed-criticality setting, each task has different demand bound functions for different criticality modes. We show that a task’s demand bound functions for different modes are inherently connected, and that we can shift demand from one function to another by tuning the parameters of the task, specifically the relative deadline.

We are free to tune the relative deadlines of tasks as long as they are never larger than the true relative deadlines that are specified by the system designer. By such tuning we can shape the demand characteristics of a task set to match the available supply of the computing platform, specified using supply bound functions (Mok et al., 2001). We present efficient algorithms that automatically shape the demand of a task set in this manner.

The standard mixed-criticality task model, which is used in most prior work, is generalized to allow arbitrary changes in task parameters between criticality modes. The generalized model also enables the addition of tasks in higher criticality modes (e.g., to implement hardware functionality in software in case of hardware faults). The manner in which a system can switch between different criticality modes is expressed with any directed acyclic graph, giving the system designer the tools necessary to express orthogonal criticality dimensions in a single system. To the best of our knowledge, systems with non-linearly ordered criticality modes have not been considered before. The adaptation of all results to the generalized model is the main new contribution of this paper, which extends a preliminary version (Ekberg and Yi, 2012).

Experimental evaluations indicate that, for most settings, the acceptance ratio of randomly generated task sets is higher with this scheduling approach than with previous approaches from the literature.

Because we allow the supply of the computing platform to be specified with supply bound functions, this scheduling approach directly enables the use of mixed-criticality scheduling within common hierarchical scheduling frameworks that employ such abstractions.

1.1 Related Work

Vestal (2007) extended fixed-priority response-time analysis of sporadic tasks to the mixed-criticality setting. His work can be considered the first on mixed-criticality scheduling. Response-time analysis for fixed-priority scheduling has since been improved by Baruah et al. (2011c)

A number of papers have considered the more restricted problem of scheduling a finite set of mixed-criticality jobs (e.g., Baruah et al., 2010, 2011a). It has been shown by Baruah et al. (2011a) that the problem of deciding whether a given set of jobs is schedulable by an optimal scheduling algorithm is NP-hard in the strong sense. Work on mixed-criticality scheduling has since been focused on finding scheduling strategies that, while being suboptimal, still work well in practice.
One of the strategies developed for scheduling a finite set of mixed-criticality jobs is the own criticality based priority (OCBP) strategy by Baruah et al. (2010). It assigns priorities to the individual jobs using a variant of the so-called Audsley approach (Audsley, 2001). This scheduling strategy was later extended by Li and Baruah (2010) to systems of mixed-criticality sporadic tasks, where priorities are calculated and assigned to all jobs in a busy period. A problem with this approach is that some runtime decisions by the scheduler are computationally very demanding. This was mitigated to some degree by Guan et al. (2011), who presented an OCBP-based scheduler for sporadic task sets where runtime decisions are of at most polynomial complexity.

An EDF-based approach called EDF-VD for scheduling implicit-deadline mixed-criticality sporadic task sets was proposed by Baruah et al. (2011b). An improvement to the schedulability analysis for EDF-VD was later described by Baruah et al. (2012). In EDF-VD, smaller (virtual) relative deadlines are used in lower criticality modes to ensure schedulability across mode changes, similar to how EDF is used in this paper. There are important differences in how relative deadlines are assigned in EDF-VD and in this paper: EDF-VD applies a single scaling factor to the relative deadlines of all tasks, and we allow them to be set independently. The main difference lies, however, in the schedulability analysis: EDF-VD uses a schedulability test based on the utilization metric, while we formulate demand bound functions. We believe that schedulability analysis based on demand bound functions is typically more precise, and is easier to generalize to more complex system models. The former is supported by the evaluation in Section 8 and the latter is supported in part by the fact that we have adapted our solution to a generalized system model in this paper.

An alternative mixed-criticality system model, which lets tasks’ periods change between criticality modes instead of their execution-time budgets, was proposed by Baruah (2012). He also provided a schedulability analysis for EDF-based scheduling of such tasks. This system model can be encoded as a special case of the generalized mixed-criticality system model described in this paper, as will be shown in Section 8.2.

Mixed-criticality scheduling on multiprocessors has been considered by Li and Baruah (2012), who combined results from the uniprocessor scheduling of EDF-VD with global EDF-based schedulability analysis of regular multiprocessor systems. Pathan (2012) instead combined ideas from fixed-priority response-time analysis for uniprocessor mixed-criticality scheduling with regular response-time analysis for fixed-priority multiprocessor scheduling.
2 Preliminaries

2.1 Simple System Model and Notation

In the first part of this paper we use the same system model as in most previous work on the scheduling of mixed-criticality tasks (e.g., Li and Baruah, 2010; Guan et al., 2011; Baruah et al., 2011c; Vestal, 2007; Baruah et al., 2011b, 2012). This is a straightforward extension of the classic sporadic task model (Mok, 1983) to a mixed-criticality setting, allowing worst-case execution times to vary between criticality levels.\footnote{The cited works differ in the assumption of implicit, constrained or arbitrary deadlines.} Formally, each such task $\tau_i$ in a mixed-criticality sporadic task set $\tau = \{\tau_1, \ldots, \tau_k\}$ is defined by a tuple $(C_i(\text{LO}), C_i(\text{HI}), D_i, T_i, L_i)$, where:

- $C_i(\text{LO}), C_i(\text{HI}) \in \mathbb{N}_+$ are the task’s worst-case execution time budgets in low- and high-criticality mode, respectively,
- $D_i \in \mathbb{N}_+$ is its relative deadline,
- $T_i \in \mathbb{N}_+$ is its minimum inter-release separation time (also called period),
- $L_i \in \{\text{LO, HI}\}$ is the criticality of the task.

We assume constrained deadlines and also make the standard assumptions about the relations between low- and high-criticality worst-case execution times:

$$\forall \tau_i \in \tau : C_i(\text{LO}) \leq C_i(\text{HI}) \leq D_i \leq T_i$$

We will generalize the above model in Section 5. In the generalized model, all task parameters, including relative deadlines and periods, can change between criticality levels. It also allows the addition of new tasks in higher criticality modes and the use of an arbitrary number of modes that are structured as any directed acyclic graph.

Let $\text{LO}(\tau) \overset{\text{def}}{=} \left\{ \tau_i \in \tau \mid L_i = \text{LO} \right\}$ denote the subset of low-criticality tasks in $\tau$, and $\text{HI}(\tau) \overset{\text{def}}{=} \left\{ \tau_i \in \tau \mid L_i = \text{HI} \right\}$ the subset of high-criticality tasks. We define low- and high-criticality utilization as

$$U_{\text{LO}}(\tau_i) \overset{\text{def}}{=} \frac{C_i(\text{LO})}{T_i}$$
$$U_{\text{HI}}(\tau_i) \overset{\text{def}}{=} \frac{C_i(\text{HI})}{T_i}$$
$$U_{\text{LO}}(\tau) \overset{\text{def}}{=} \sum_{\tau_i \in \tau} U_{\text{LO}}(\tau_i)$$
$$U_{\text{HI}}(\tau) \overset{\text{def}}{=} \sum_{\tau_i \in \text{HI}(\tau)} U_{\text{HI}}(\tau_i).$$

For compactness of presentation we use the notation $\llbracket \cdot \rrbracket_c$ and $\llbracket \cdot \rrbracket^c$ to constrain an expression from below or above, such that $\llbracket A \rrbracket_c \overset{\text{def}}{=} \max(A, c)$ and $\llbracket A \rrbracket^c \overset{\text{def}}{=} \min(A, c)$. Also, $\llbracket A \rrbracket^c_c \overset{\text{def}}{=} \llbracket \llbracket A \rrbracket_c \rrbracket^c$. The semantics of the system model is as follows. The system starts in low-criticality mode, and as long as it remains there, each task $\tau_i \in \tau$ releases...
a (possibly infinite) sequence of jobs $\langle J_1^i, J_2^i, \ldots \rangle$ in the standard way for sporadic tasks: if $r(J), d(J) \in \mathbb{R}$ are the release time and deadline of job $J$, then

- $r(J_{i}^{k+1}) \geq r(J_{i}^{k}) + T_{i}$,
- $d(J_{i}^{k}) = r(J_{i}^{k}) + D_{i}$.

The time interval $[r(J), d(J)]$ is called the scheduling window of job $J$. If any job executes for its entire low-criticality worst-case execution time budget without signaling that it has finished, the system will immediately switch to high-criticality mode. This switch signifies that the system’s behavior is not consistent with the assumptions made at the lower level of assurance (in particular, the worst-case execution time estimates are invalid). After the switch we are not required to meet any deadlines for low-criticality jobs, but we must still meet all deadlines for high-criticality jobs, even if they execute for up to their high-criticality worst-case execution times (i.e., the high-criticality tasks get increased execution-time budgets). In practice, the low-criticality jobs can continue to execute whenever the processor would otherwise be idle, but from the modeling perspective we simply view all low-criticality tasks in $\text{LO}(\tau)$ as being discarded along with their active jobs at the time of the switch. The tasks in $\text{HI}(\tau)$ carry on unaffected. If the system has switched to high-criticality mode, it will never switch back to low-criticality.$^2$

For such a system to be successfully scheduled, all (non-discarded) jobs must always meet their deadlines. Note that the only jobs that exist in high-criticality mode are from tasks in $\text{HI}(\tau)$. Since low-criticality jobs do not run in high-criticality mode, we omit to specify high-criticality worst-case execution times for low-criticality tasks.

**Example 2.1.** As a running example we will use the following simple task set. It consists of three tasks ($\tau_1$, $\tau_2$ and $\tau_3$), one of low- and two of high-criticality:

<table>
<thead>
<tr>
<th>Task</th>
<th>C(LO)</th>
<th>C(HI)</th>
<th>D</th>
<th>T</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>$\text{LO}$</td>
<td></td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>$\text{HI}$</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>$\text{HI}$</td>
</tr>
</tbody>
</table>

This task set is not schedulable by any fixed-priority scheduler on a dedicated unit-speed processor, as can be verified by trying all 6 possible priority assignments. We can also see that the task set is not schedulable directly by EDF: in the scenario where all tasks release a job at the same time, EDF would execute $\tau_1$ first, leaving $\tau_2$ and $\tau_3$ unable to finish on time if they need to execute for $C_2(\text{HI})$ and $C_3(\text{HI})$, respectively. Neither does the task set pass the schedulability tests for OCBP (Li and Baruah, 2010; Guan et al., 2011) or EDF-VD (Baruah et al., 2011b, 2012), even if deadlines are increased to.

$^2$One could easily find a time point where it is safe to switch back, e.g., at any time the system is idle, but it is out of scope of this paper.
be implicit, as is required by EDF-VD. However, we will see that its demand characteristics can be tuned using the techniques presented in this paper until it is schedulable by EDF.

2.2 Demand Bound Functions

A successful approach to analyzing the schedulability of real-time workloads is to use demand bound functions (Baruah et al., 1990). A demand bound function captures the maximum execution demand of a task in any time interval of a given size.

**Definition 2.1** (Demand bound function). A demand bound function \( \text{dbf}(\tau_i, \ell) \) gives an upper bound on the maximum possible execution demand of task \( \tau_i \) in any time interval of length \( \ell \), where demand is calculated as the total amount of required execution time of jobs with their whole scheduling windows within the time interval.

There exist methods for precisely computing the demand bound functions for many popular task models in the normal (non-mixed-criticality) setting. For example, the demand bound function for a given \( \ell \) can be computed in constant time for a standard sporadic task (Baruah et al., 1990).

A similar concept is the supply bound function \( \text{sbf}(\ell) \) (Mok et al., 2001), which lower bounds the amount of supplied execution time of the platform in any time window of size \( \ell \). For example, a unit-speed, dedicated uniprocessor has \( \text{sbf}(\ell) = \ell \). Other platforms, such as virtual servers used in hierarchical scheduling, have their own particular supply bound functions (e.g., Mok et al., 2001; Shin and Lee, 2003). We say that a supply bound function \( \text{sbf} \) is of at most unit speed if

\[
\text{sbf}(0) = 0 \quad \land \quad \forall \ell, k \geq 0 : \text{sbf}(\ell + k) - \text{sbf}(\ell) \leq k.
\]

We assume that a supply bound function is linear in all intervals \([k, k + 1]\) between consecutive integer points \(k\) and \(k + 1\). The assumption of piecewise-linear supply bound functions is a natural one, and to the best of our knowledge, all proposed virtual resource platforms in the literature have such supply bound functions.

The key insight that make demand and supply bound functions useful for the analysis of real-time systems is the following known fact.

**Proposition 2.1** (e.g., see Shin and Lee (2003)). A non-mixed-criticality task set \( \tau \) is successfully scheduled by the earliest deadline first (EDF) algorithm on a (uniprocessor) platform with supply bound function \( \text{sbf} \) if

\[
\forall \ell \geq 0 : \sum_{\tau_i \in \tau} \text{dbf}(\tau_i, \ell) \leq \text{sbf}(\ell).
\]
3 Demand Bound Functions for Mixed-Criticality Tasks

We extend the idea of demand bound functions to the mixed-criticality setting. For each task we will construct two demand bound functions, \( \text{dbf}_{\text{LO}} \) and \( \text{dbf}_{\text{HI}} \), for the low- and high-criticality modes, respectively. Proposition 2.1 is extended in the straightforward way:

**Proposition 3.1.** A mixed-criticality task set \( \tau \) is schedulable by EDF on a platform with supply bound function \( \text{sbf}_{\text{LO}} \) in low-criticality mode and \( \text{sbf}_{\text{HI}} \) in high-criticality mode if both of the following conditions hold:

- **Condition \( S_{\text{LO}} \):** \( \forall \ell \geq 0 : \sum_{\tau_i \in \tau} \text{dbf}_{\text{LO}}(\tau_i, \ell) \leq \text{sbf}_{\text{LO}}(\ell) \)
- **Condition \( S_{\text{HI}} \):** \( \forall \ell \geq 0 : \sum_{\tau_i \in \text{Hi}(\tau)} \text{dbf}_{\text{HI}}(\tau_i, \ell) \leq \text{sbf}_{\text{HI}}(\ell) \)

Conditions \( S_{\text{LO}} \) and \( S_{\text{HI}} \) capture the schedulability of the task set in low- and high-criticality mode. While the two modes can be analyzed separately with the above conditions, we will see that the demand in high-criticality mode depends on what can happen in low-criticality mode.

We assume, without loss of generality, that \( \text{sbf}_{\text{LO}} \) is of at most unit speed. This can always be achieved by simply scaling the parameters of the task set together with \( \text{sbf}_{\text{LO}} \) and \( \text{sbf}_{\text{HI}} \). Note that \( \text{sbf}_{\text{LO}} \) and \( \text{sbf}_{\text{HI}} \) may be different, allowing a change of processor speed or virtual server scheduling policy when switching to high-criticality mode.

How then do we construct these demand bound functions? In the case of \( \text{dbf}_{\text{LO}} \) it is simple. In low-criticality mode, each task \( \tau_i \) behaves like a normal sporadic task, and all of its jobs are guaranteed to execute for at most \( C_i(\text{LO}) \) time units (otherwise the system, by definition, would switch to high-criticality mode). We can therefore use the standard method for computing demand bound functions for sporadic tasks (Baruah et al., 1990). With \( \text{dbf}_{\text{HI}} \) it gets more tricky because we need to consider the high-criticality jobs that are active during the switch to high-criticality mode.

**Definition 3.1 (Carry-over jobs).** A job from a high-criticality task that is active (released, but not finished) at the time of the switch to high-criticality mode is called a **carry-over job**.

### 3.1 Characterizing the Demand of Carry-Over Jobs

In high-criticality mode we need to finish the remaining execution time of carry-over jobs before their respective deadlines. The demand of carry-over jobs must therefore be accounted for in each high-criticality task’s \( \text{dbf}_{\text{HI}} \). Conceptually, when analyzing the schedulability in high-criticality mode, we can think of a carry-over job as a job that is released at the time of the switch.
Figure 1. After a switch to high-criticality mode, the remaining execution demand of a carry-over job must be finished in its remaining scheduling window.

However, the scheduling window of such a job is the remaining interval between switch and deadline (see Fig. 1), and can therefore be shorter than for other jobs of the same task. Because it might have executed for some time before the switch, its execution demand may also be lower.

For the sake of bounding the demand in high-criticality mode (in order to meet Condition $S_{HI}$), we can assume that the demand is met in low-criticality mode (Condition $S_{LO}$), or the task set would be deemed unschedulable anyway. In other words, we seek to show $S_{LO} \land S_{HI}$ by showing $S_{LO} \land (S_{LO} \rightarrow S_{HI})$. For a system scheduled by EDF, we can therefore assume that all deadlines are met in low-criticality mode when we bound the demand in high-criticality mode.

Consider then what we can show about the remaining execution demand of carry-over jobs. At the time of the switch to high-criticality mode, a carry-over job from high-criticality task $\tau_i$ has $x$ time units left until its deadline, for some $x \geq 0$. The remaining scheduling window of this job is therefore of length $x$. Since this job would have met its deadline in low-criticality mode if the switch had not happened, there can be at most $x$ time units left of its low-criticality execution demand $C_i(LO)$ at the time of the switch (this follows directly from the assumption that $sbf_{LO}$ is of at most unit speed). The job must therefore have executed for at least $\lceil C_i(LO) - x \rceil_0$ time units before the switch. Since the system has switched to high-criticality mode, the job may now execute for up to $C_i(HI)$ time units in total. The total execution demand remaining for the carry-over job after the switch is therefore at most $C_i(HI) - \lceil C_i(LO) - x \rceil_0$. Unfortunately, as $x$ becomes smaller, this demand is increasingly difficult to accommodate, and leads to $dbf_{HI}(\tau_i, 0) = C_i(HI) - C_i(LO)$ in the extreme case. Clearly, with such bounds we cannot hope to satisfy Condition $S_{HI}$.

Next we will show how this problem can be mitigated.
3.2 Adjusting the Demand of Carry-Over Jobs

The problem above stems from the fact that EDF may execute a high-criticality job quite late in low-criticality mode. When the system switches to high-criticality mode, a carry-over job can be left with a very short scheduling window in which to finish what remains of its high-criticality worst-case execution demand. In order to increase the size of the remaining scheduling window we separate the relative deadlines used in the different modes. For a task $\tau_i$ we let EDF use relative deadlines $D_i(\text{LO})$ and $D_i(\text{HI})$, such that if a job is released at time $t$, the priority assigned to it by EDF is based on the value $t + D_i(\text{LO})$ in low-criticality mode and based on $t + D_i(\text{HI})$ in high-criticality mode. This is essentially the same run-time scheduling as that of EDF-VD (Baruah et al., 2011b, 2012).

We can safely lower the relative deadline of a task because meeting the earlier deadline implies meeting the original (true) deadline. We can gain valuable extra slack time for a carry-over job from high-criticality task $\tau_i$ by lowering $D_i(\text{LO})$, albeit at the cost of a worsened demand in low-criticality mode. We therefore want $D_i(\text{LO}) = D_i$ if $L_i = \text{LO}$ and $D_i(\text{LO}) \leq D_i(\text{HI}) = D_i$ if $L_i = \text{HI}$. Also, $C_i(\text{LO}) \leq D_i(\text{LO})$ is assumed, just as with the original deadline. Note that $D_i(\text{LO})$ is not an actual relative deadline for $\tau_i$ in the sense that it does not necessarily correspond to the timing constraints specified by the system designer. However, it is motivated to call it a “deadline”, because we construct each $\text{dbf}_{\text{LO}}$ and use EDF in low-criticality mode as if it was the relative deadline. With separated relative deadlines we can make stronger guarantees about the remaining execution demand of carry-over jobs:

**Lemma 3.2 (Demand of carry-over jobs).** Assume that EDF uses relative deadlines $D_i(\text{LO})$ and $D_i(\text{HI})$ with $D_i(\text{LO}) \leq D_i(\text{HI}) = D_i$ for high-criticality task $\tau_i$, and that we can guarantee that the demand is met in low-criticality mode (using $D_i(\text{LO})$). If the switch to high-criticality mode happens when a job from $\tau_i$ has a remaining scheduling window of $x$ time units left until its true deadline, as illustrated in Fig. 2, then the following hold:

1. If $x < D_i(\text{HI}) - D_i(\text{LO})$, then the job has already finished before the switch.

2. If $x \geq D_i(\text{HI}) - D_i(\text{LO})$, then the job may be a carry-over job, and no less than $\lfloor C_i(\text{LO}) - x + D_i(\text{HI}) - D_i(\text{LO}) \rfloor_0$ time units of the job’s work were finished before the switch.

**Proof.** In the first case, the switch to high-criticality mode happens after the low-criticality deadline. Since we assume that the demand is met in low-criticality mode (using relative deadline $D_i(\text{LO})$), EDF is guaranteed to finish the job by this deadline, and therefore it was finished by the time of the switch.

In the second case, there are $x - (D_i(\text{HI}) - D_i(\text{LO}))$ time units left until the low-criticality deadline. Since the demand is guaranteed to be met in low-criticality mode, and the supply of the platform is of at most unit speed, there...
can be at most \( x - (D_i(HI) - D_i(LO)) \) time units left of the job’s low-criticality execution demand. At least \( \lceil C_i(LO) - x + D_i(HI) - D_i(LO) \rceil_0 \) time units of the job’s work must therefore have been finished already by the time of the switch.

\[ x = D_i(HI) - D_i(LO) \]

**Figure 2.** A carry-over job of \( \tau_i \) has a remaining scheduling window of length \( x \) after the switch to high-criticality mode. Here the switch happens before the job’s low-criticality deadline.

Next we will show how to define \( \text{dbf}_{LO}(\tau_i, \ell) \) and \( \text{dbf}_{HI}(\tau_i, \ell) \) for a given \( D_i(LO) \). An algorithm for computing reasonable values for \( D_i(LO) \) for each task \( \tau_i \in \tau \) is presented in Section 4.

### 3.3 Formulating the Demand Bound Functions

As described above, while the system is in low-criticality mode, each task \( \tau_i \) behaves as a normal sporadic task with parameters \( C_i(LO), D_i(LO) \) and \( T_i \). Note that it uses relative deadline \( D_i(LO) \), where \( D_i(LO) = D_i \) if \( L_i = LO \) and \( D_i(LO) \leq D_i(HI) = D_i \) if \( L_i = HI \). A tight demand bound function of such a task is known (Baruah et al., 1990):

\[
\text{dbf}_{LO}(\tau_i, \ell) = \left\lceil \left( \frac{\ell - D_i(LO)}{T_i} \right) + 1 \right\rceil \cdot C_i(LO) \]

(1)

The demand bound function for task \( \tau_i \) in high-criticality mode, \( \text{dbf}_{HI}(\tau_i, \ell) \), must provide an upper bound on the maximum execution demand of jobs from \( \tau_i \) with scheduling windows inside any interval of length \( \ell \). This may include one carry-over job. From Lemma 3.2 we know that the (remaining) scheduling window of a carry-over job from \( \tau_i \) is at least \( D_i(HI) - D_i(LO) \) time units long. A time interval of length \( D_i(HI) - D_i(LO) \) is therefore the smallest in which we can fit the scheduling window of *any* job from \( \tau_i \). More generally, the smallest time interval in which we can fit the scheduling windows of \( k \) jobs...
is of length $(D_i(\text{HI}) - D_i(\text{LO})) + (k - 1) \cdot T_i$. The execution demand of $\tau_i$ in an interval of length $\ell$ is therefore bounded by

$$\text{full}_{\text{HI}}(\tau_i, \ell) \overset{\text{def}}{=} \left\lceil \left( \frac{\ell - (D_i(\text{HI}) - D_i(\text{LO}))}{T_i} \right) + 1 \right\rceil \cdot C_i(\text{HI})$$

The function $\text{full}_{\text{HI}}(\tau_i, \ell)$ is disregarding that a carry-over job may have finished some execution in low-criticality mode (i.e., it is counting $C_i(\text{HI})$ for all jobs). We can check whether all jobs that contributed execution demand to $\text{full}_{\text{HI}}(\tau_i, \ell)$ can fit their scheduling windows into an interval of length $\ell$ without one of them being a carry-over job. If one must be a carry-over job, we can subtract the execution time that it must have finished before the switch according to Lemma 3.2.

Switch to high-criticality mode

![Figure 3](image_url)

*Figure 3.* After fitting a number of full jobs into an interval of length $\ell$, there are $\ell \mod T_i$ time units left for either another full job, a carry-over job, or no job at all. In this figure it is enough for a carry-over job.

As shown in Fig. 3, for a time interval of length $\ell$, there are at most $x = \ell \mod T_i$ time units left for the “first” job (which may be a carry-over job). If $x \geq D_i(\text{HI})$, it is enough for the scheduling window of a full job, and we cannot subtract anything from $\text{full}_{\text{HI}}(\tau_i, \ell)$. If $x < D_i(\text{HI}) - D_i(\text{LO})$, all jobs that contributed to $\text{full}_{\text{HI}}(\tau_i, \ell)$ can fit their entire periods inside the interval, so there is again nothing to subtract. Otherwise, we use Lemma 3.2 to quantify the amount of work that must have been finished in low-criticality mode:

$$\text{done}_{\text{HI}}(\tau_i, \ell) \overset{\text{def}}{=} \begin{cases} \left\lceil C_i(\text{LO}) - x + D_i(\text{HI}) - D_i(\text{LO}) \right\rceil_0, & \text{if } D_i(\text{HI}) \geq x \geq D_i(\text{HI}) - D_i(\text{LO}) \\ 0, & \text{otherwise} \end{cases}$$

where $x = \ell \mod T_i$. Note that by maximizing the remaining scheduling window of the carry-over job (to $\ell \mod T_i$) we also maximize its remaining execution demand.

The two terms can now be combined to form the demand bound function in high-criticality mode:

$$\text{dbf}_{\text{HI}}(\tau_i, \ell) \overset{\text{def}}{=} \text{full}_{\text{HI}}(\tau_i, \ell) - \text{done}_{\text{HI}}(\tau_i, \ell)$$
Example 3.1. Consider task $\tau_3$ from Example 2.1. Part of the demand bound functions for $\tau_3$ are shown in Fig. 4, using two different values for $D_3(LO)$. Note that a smaller $D_3(LO)$ leads to a lessened demand in high-criticality mode, at the cost of an increased demand in low-criticality mode.

Figure 4. Demand bound functions for task $\tau_3$ from Example 2.1 with two different values for $D_3(LO)$.

4 Tuning Relative Deadlines

In the previous section we constructed demand bound functions for mixed-criticality sporadic tasks, where the relative deadlines used by EDF may differ in low- and high-criticality mode for high-criticality tasks. The motivation for separating the relative deadlines used is that by artificially lowering the relative deadline $D_i(LO)$ used in low-criticality mode, we can lessen $\tau_i$'s demand in high-criticality mode at the cost of increasing the demand in low-criticality mode. By choosing suitable values for $D_i(LO)$ for all tasks $\tau_i \in HI(\tau)$, we are increasing our chances of fitting the total demand under the guaranteed supply in both modes, and thereby make both Conditions $S_{LO}$ and $S_{HI}$ of Proposition 3.1 hold.
We are constrained to pick a value for $D_i(\text{LO})$ such that $C_i(\text{LO}) \leq D_i(\text{LO}) \leq D_i$. This gives us

$$\prod_{\tau_i \in \text{HI}(\tau)} (D_i - C_i(\text{LO}) + 1)$$

possible combinations for the task set. The number of combinations is exponentially increasing with the number of high-criticality tasks, and it is infeasible to simply try all combinations. We instead seek a heuristic algorithm for tuning the relative deadlines of all tasks. In this section we present one such algorithm, which is of pseudo-polynomial time complexity for suitable supply bound functions.

The following lemma is a key insight for understanding the effects of changing relative deadlines. A proof is given in Appendix A.

**Lemma 4.1** (Shifting). If high-criticality tasks $\tau_i$ and $\tau_j$ are identical (i.e., have equal parameters), except that $D_i(\text{LO}) = D_j(\text{LO}) - \delta$ for $\delta \in \mathbb{Z}$, then

$$dbf_{\text{LO}}(\tau_i, \ell) = dbf_{\text{LO}}(\tau_j, \ell + \delta)$$

$$dbf_{\text{HI}}(\tau_i, \ell) = dbf_{\text{HI}}(\tau_j, \ell - \delta)$$

In other words, if we consider the demand bound functions graphically as in Fig. 4, then by decreasing $D_i(\text{LO})$ by $\delta$, we are allowed to move $dbf_{\text{HI}}(\tau_i, \ell)$ by $\delta$ steps to the right at the cost of moving $dbf_{\text{LO}}(\tau_i, \ell)$ by $\delta$ steps to the left. Informally, we can think of the problem as moving around the $dbf_{\text{LO}}$ and $dbf_{\text{HI}}$ of each task until we hopefully find a configuration where the total demand of the task set is met by the supply in both low- and high-criticality mode.

Algorithm 1 tunes the demand of a task set in a somewhat greedy fashion. Let $s_{\text{LO}}(\ell)$ and $s_{\text{HI}}(\ell)$ be predicates corresponding to the inequalities found in Conditions $S_{\text{LO}}$ and $S_{\text{HI}}$, respectively:

$$s_{\text{LO}}(\ell) \overset{\text{def}}{=} \sum_{\tau_i \in \tau} dbf_{\text{LO}}(\tau_i, \ell) \leq sbf_{\text{LO}}(\ell)$$

$$s_{\text{HI}}(\ell) \overset{\text{def}}{=} \sum_{\tau_i \in \text{HI}(\tau)} dbf_{\text{HI}}(\tau_i, \ell) \leq sbf_{\text{HI}}(\ell)$$

The general idea is to check $s_{\text{LO}}(\ell)$ and $s_{\text{HI}}(\ell)$ for increasing time interval lengths $\ell$ (from 0 up to an upper bound $\ell_{\text{max}}$ described in Section 4.1). As soon as it finds a value for $\ell$ for which either condition fails, it changes one relative deadline (or terminates) and goes back to $\ell = 0$:

- If $s_{\text{HI}}(\ell)$ fails, the low-criticality relative deadline of one task is decreased by 1. It picks the task $\tau_i$ which would see the largest decrease in $dbf_{\text{HI}}(\tau_i, \ell)$ when $D_i(\text{LO})$ is decreased by 1 (ties broken arbitrarily).
- If $s_{\text{LO}}(\ell)$ fails, the latest deadline change is undone. If there is no change to undo, the algorithm fails. Note that it backtracks at most one step in this way.
Algorithm 1: GreedyTuning(τ)

begin

candidates ← {i | τi ∈ HI(τ)}
mod ← ⊥
ℓmax ← upper bound for ℓ in Conditions SLO and SHI
repeat

changed ← false
for ℓ = 0, 1, ..., ℓmax do

if ¬sLO(ℓ) then
    if mod = ⊥ then
        return FAILURE
    Dmod(LO) ← Dmod(LO) + 1
    candidates ← candidates \ {mod}
    mod ← ⊥
    changed ← true
    break
else if ¬sHI(ℓ) then
    if candidates = ∅ then
        return FAILURE
    mod ← arg max,i∈candidates (dbfHI(τi, ℓ) − dbfHI(τi, ℓ − 1))
    Dmod(LO) ← Dmod(LO) − 1
    if Dmod(LO) = Cmod(LO) then
        candidates ← candidates \ {mod}
    changed ← true
    break
until ¬changed
return SUCCESS

The algorithm returns SUCCESS only if it has found low-criticality relative deadlines with which sLO(ℓ) and sHI(ℓ) hold for all ℓ ∈ {0, 1, ..., ℓmax}. This implies that both Conditions SLO and SHI hold, as will be shown in Section 4.1. Therefore, the algorithm terminates with SUCCESS only if the task set is schedulable according to Proposition 3.1. If the algorithm terminates with FAILURE, it has failed to find relative deadlines with which both Conditions SLO and SHI hold. This does not necessarily mean that such relative deadlines can not be found in some other way.

Example 4.1. Consider how Algorithm 1 assigns values to D2(LO) and D3(LO) for the two high criticality tasks τ2 and τ3 in the task set from Example 2.1. We assume a dedicated platform (sbfLO(ℓ) = sbfHI(ℓ) = ℓ). Fig. 5 shows the
demand bound functions for this task set with unmodified relative deadlines. In the first iteration, \( s_{\text{HI}}(0) \) fails, and \( D_3(\text{LO}) \) is decreased by 1. In the second iteration, \( s_{\text{HI}}(0) \) fails again, but this time \( D_2(\text{LO}) \) is decreased by 1. In the third iteration, \( s_{\text{HI}}(1) \) fails and \( D_3(\text{LO}) \) is decreased by 1 again. This is then repeated two more times where \( s_{\text{HI}}(\ell) \) fails at \( \ell = 2 \) and \( \ell = 3 \), respectively, and \( D_3(\text{LO}) \) is lowered two more times. Both \( s_{\text{LO}}(\ell) \) and \( s_{\text{HI}}(\ell) \) then hold for all \( \ell \in \{0, 1, \ldots, \ell_{\text{max}}\} \), and the algorithm terminates with \( D_2(\text{LO}) = 5 \) and \( D_3(\text{LO}) = 2 \), resulting in the demand bound functions shown in Fig. 6. \( \square \)

![Demand bound functions for the tasks from Example 2.1 with unmodified low-criticality relative deadlines \((D_i(\text{LO}) = D_i(\text{HI}) = D_i)\).](image)

**Figure 5.** Demand bound functions for the tasks from Example 2.1 with unmodified low-criticality relative deadlines \((D_i(\text{LO}) = D_i(\text{HI}) = D_i)\).

### 4.1 Complexity and Correctness of the Algorithm

For the complexity of Algorithm 1, note that each \( \tau_i \in \text{HI}(\tau) \) will have its deadline \( D_i(\text{LO}) \) changed at most \( D_i - C_i(\text{LO}) + 1 \) times. In every iteration of the outer loop some low-criticality relative deadline is changed, or the algorithm terminates, so the outer loop is iterated at most

\[
\sum_{\tau_i \in \text{HI}(\tau)} (D_i - C_i(\text{LO}) + 1)
\]

times. The inner for-loop is iterated at most \( \ell_{\text{max}} + 1 \) times for every iteration of the outer loop. The algorithm is therefore of pseudo-polynomial time com-
Figure 6. Demand bound functions for the tasks from Example 2.1 after having low-criticality relative deadlines tuned by Algorithm 1.

complexity if \( \ell_{\max} \) is pseudo-polynomial. We will see that a pseudo-polynomial \( \ell_{\max} \) can be found in the common setting where the supply is from a dedicated platform.

The algorithm terminates with SUCCESS only if it has found relative deadlines with which both \( s_{LO}(\ell) \) and \( s_{HI}(\ell) \) hold for all \( \ell \in \{0, 1, \ldots, \ell_{\max}\} \). However, in Proposition 3.1, the inequalities \( s_{LO}(\ell) \) and \( s_{HI}(\ell) \) should hold for all \( \ell \geq 0 \). We will show here that \( \ell_{\max} \) can be found such that if \( s_{LO}(\ell) \) and \( s_{HI}(\ell) \) hold for \( \ell \in \{0, 1, \ldots, \ell_{\max}\} \), then they hold for all \( \ell \geq 0 \).

Consider first why it is enough to check only integer-valued \( \ell \). Both \( sbf_{LO} \) and \( sbf_{HI} \) are linear in all intervals \([k, k+1]\) between consecutive integer points \( k \) and \( k+1 \). All \( dbf_{LO} \) and \( dbf_{HI} \) are non-decreasing in \( \ell \) and also linear in all intervals \([k, k+1]\) for consecutive integers \( k \) and \( k+1 \) (and so are the left-hand sides of \( s_{LO}(\ell) \) and \( s_{HI}(\ell) \)). It follows directly that if \( s_{LO}(\ell) \) or \( s_{HI}(\ell) \) does not hold for an \( \ell \in [k, k+1] \) with \( k \in \mathbb{N} \), then it also does not hold for either \( k \) or \( k+1 \).

How a bound \( \ell_{\max} \) can be found depends on the supply bound functions used. It is always possible to use the hyperperiod as the bound \( \ell_{\max} \). However, for a dedicated uniprocessor \( (sbf_{LO}(\ell) = sbf_{HI}(\ell) = \ell) \) we can use established methods (Baruah et al., 1990) to calculate a pseudo-polynomial \( \ell_{\max} \) as long as \( U_{LO}(\tau) \) and \( U_{HI}(\tau) \) are a priori bounded by a constant smaller than 1. To see
this, we first create mappings \( f_{LO} \) and \( f_{HI} \) from mixed-criticality sporadic tasks to normal (non-mixed-criticality) sporadic tasks \((C, D, T)\) in the following way:

\[
\begin{align*}
  f_{LO}(\tau_i) &\overset{\text{def}}{=} (C_i(LO), D_i(LO), T_i) \\
  f_{HI}(\tau_i) &\overset{\text{def}}{=} (C_i(HI), D_i(HI) - D_i(LO), T_i)
\end{align*}
\]

Note that with the demand bound function \( \text{dbf} \) for normal sporadic tasks, first described by Baruah et al. (1990), we have

\[
\text{dbf}(f_{LO}(\tau_i), \ell) = \text{dbf}_{LO}(\tau_i, \ell)
\]

and

\[
\text{dbf}(f_{HI}(\tau_i), \ell) \geq \text{dbf}_{HI}(\tau_i, \ell) \geq \text{dbf}_{HI}(\tau_i, \ell).
\]

Also, if \( U \) gives the utilization of a normal sporadic task, we have

\[
U(f_{LO}(\tau_i)) = U_{LO}(\tau_i)
\]

and

\[
U(f_{HI}(\tau_i)) = U_{HI}(\tau_i).
\]

Baruah et al. (1990) showed how to construct a pseudo-polynomial bound for normal sporadic task sets such that the inequality in Proposition 2.1 holds for all \( \ell \) larger than the bound (using a dedicated uniprocessor), as long as the utilization of the task set is bounded by a constant smaller than 1. Clearly, if we construct such a bound \( \ell_{\max}^{LO} \) for the task set \( \{f_{LO}(\tau_i) \mid \tau_i \in \tau\} \), it is also valid for Condition \( S_{LO} \) in Proposition 3.1. Similarly, such a bound \( \ell_{\max}^{HI} \) for the task set \( \{f_{HI}(\tau_i) \mid \tau_i \in HI(\tau)\} \) is valid for Condition \( S_{HI} \) of Proposition 3.1. We can therefore use \( \ell_{\max} = \max(\ell_{\max}^{LO}, \ell_{\max}^{HI}) \) for Algorithm 1.3

5 Generalizing the Mixed-Criticality Task Model

In Section 2 we described the standard mixed-criticality sporadic task model, which is used in most previous work on mixed-criticality scheduling (e.g., Li and Baruah, 2010; Guan et al., 2011; Baruah et al., 2011c; Vestal, 2007; Baruah et al., 2011b, 2012). This task model is execution-time centric, as it focuses solely on differences in the worst-case execution-time parameter between criticality levels. Arguably, if one has to pick a single parameter to focus on, the execution time is a good choice because it is almost always an approximation, and its value typically varies greatly with the level of assurance that is desired. There are cases where it is desirable to vary other parameters, though. Consider, for example, a task that is triggered by external events. The period of such a task should be an under-approximation of the time interval between two consecutive trigger events. At different criticality levels, different values for the period parameter might be more suitable, depending on the required assurance that it is a safe under-approximation. Baruah (2012) introduced a task model where the period parameter differs between criticality levels, instead of the execution-time parameter.

\[3\text{A small technical issue is that the bound by Baruah et al. (1990) is dependent on the relative deadlines of tasks, which are changed by Algorithm 1. The issue is easily avoided by using the largest bound generated with any of the possible relative deadlines that may be assigned (this is simply } D_i(LO) = C_i(LO) \text{ for all } \tau_i \in HI(\tau)). \text{ An even easier solution is to use an alternative bound that is independent of relative deadlines, e.g., the one described by Stigge et al. (2011).} \]
We would like the task model to be as general as possible, without forcing an interpretation of it on the system designer. It should be up to the system designer to decide what it means for the system to be in any one particular criticality mode, e.g., which tasks should run there; what parameters they should have; and which events trigger the system to switch to or from that criticality mode, be it an execution-time budget overrun, a hardware malfunction or anything else. Note that such generalizations bring the notion of mixed criticality closer to that of regular mode switches (e.g., see Real and Crespo, 2004). We think that this is a proper development, as long as we retain the differences between mixed criticality on the one hand, and regular mode switches on the other. We argue that the most important difference between these concepts is that while a regular mode switch often is controlled, a change of criticality modes is forced upon the system by immediate and unexpected events. Such events cannot be handled by deferring task releases as is often done for controlled mode switches. Instead, the possibility of them must be prepared for in advance as is done in mixed-criticality scheduling. Still, the border between these concepts is somewhat fuzzy, and we think that it is not unlikely that some existing solutions regarding the scheduling of regular mode switching systems can be adapted for mixed-criticality scheduling. One can look at mixed-criticality systems as mode switching systems with a particular class of mode change protocols.

We will generalize the mixed-criticality sporadic task model to allow all task parameters to change between criticality modes. It will also be possible to add new tasks to the system when it switches to a higher criticality mode. To motivate the latter, consider as an example a system where a hardware malfunction triggers the creation of a new task that compensates for the missing functionality in software; in this system a hardware fault triggers a switch to a higher criticality mode, where some new tasks are added and possibly some old tasks are suspended or have their parameters changed. Another example is a distributed system where a node failure causes some critical tasks to be migrated to another node. From the point of view of the node receiving the tasks, there is a switch to a new criticality mode where it must accommodate the new tasks.

In addition, we will lift the restriction to only two criticality modes, and allow an arbitrary number of modes that are not necessarily linearly ordered. The ways in which criticality modes can be changed are expressed using any directed acyclic graph (DAG), as in Example 5.1. To the best of our knowledge, non-linearly ordered criticality modes have not been considered for mixed-criticality scheduling before.

**Example 5.1.** Consider a system that the designer wants to have different criticality levels with different worst-case execution time budgets, in the standard manner for mixed-criticality systems. Also, the designer wants to be able to compensate for missing hardware functionality in software in the case of some...
specific hardware failures, and therefore wishes to add one or more tasks and possibly modify others in the face of such an event. The criticality modes of this system could be arranged as in Fig. 7. The system would start running in the mode entitled $m_{\text{NORMAL}}$, which is its normal operating mode. In the event of an attempted execution-time overrun, it would switch to the mode $m_{\text{WCET}}$ where some non-critical tasks may be suspended, and the remaining tasks get higher worst-case execution time budgets. In the event of a hardware fault, the system instead switches to the mode $m_{\text{HW}}$, in which some new tasks are added. In order to accommodate the new tasks, the designer may wish to suspend some old ones, or lower the demand of some tasks by, for example, increasing their periods or relative deadlines. In the event that both execution-time overruns and hardware faults occur, the system switches to the mode $m_{\text{WCET+HW}}$, where the designer must decide which tasks are most critical for the system in such extreme conditions.

![Figure 7. An example structure of a system’s criticality modes.](image)

5.1 Formalizing the Generalized System Model

A generalized mixed-criticality sporadic task system is formally defined by a pair $(\tau, G)$, where $\tau$ is a set of tasks, $\{\tau_1, \ldots, \tau_k\}$, and $G$ is a DAG describing the structure of the criticality modes. The vertex set $V(G)$ contains the possible criticality modes and the edge set $E(G)$ the ways in which criticality modes may change. The graph $G$ is called the criticality-mode structure of the system.

Each task $\tau_i \in \tau$ is defined by a set $\mathcal{L}_i$, and a tuple $(C_i(m), D_i(m), T_i(m))$ for each $m \in \mathcal{L}_i$, where:

- $\mathcal{L}_i \subseteq V(G)$ is the set of criticality modes in which $\tau_i$ is active,
- $C_i(m) \in \mathbb{N}_+$ is its worst-case execution time in criticality mode $m$,
- $D_i(m) \in \mathbb{N}_+$ is its relative deadline in criticality mode $m$,
- $T_i(m) \in \mathbb{N}_+$ is its minimum inter-release separation time (also called period) in criticality mode $m$.
We assume that for each $\tau_i \in \tau$ and for each $m \in \mathcal{L}_i$ that $C_i(m) \leq D_i(m) \leq T_i(m)$, similar to the assumptions about the standard task model. However, there are no restrictions on the relations between the parameters of a task in different criticality modes, i.e., all its parameters may change to arbitrary values.

Utilization is defined in the natural way:

$$U_m(\tau_i) \overset{\text{def}}{=} \begin{cases} \frac{C_i(m)}{T_i(m)}, & \text{if } m \in \mathcal{L}_i \\ 0, & \text{otherwise} \end{cases}$$

The new model generalizes the standard mixed-criticality task model described in Section 2. Note that the criticality-mode structure $G$ for the standard model would have only two vertices, $V(G) = \{\text{LO}, \text{HI}\}$, which are connected by a single edge, $E(G) = \{(\text{LO}, \text{HI})\}$.

The semantics of the generalized model is very similar to those of the standard model: In criticality mode $m$, each task $\tau_i$ that is active in $m$ releases jobs as if it was a normal sporadic task with parameters $(C_i(m), D_i(m), T_i(m))$. The system may switch from criticality mode $m$ to $m'$ if $(m, m') \in E(G)$, where $G$ is the criticality-mode structure. If the system switches from $m$ to $m'$, each task $\tau_i$ can be affected in different ways:

- If $m \in \mathcal{L}_i$ and $m' \notin \mathcal{L}_i$, the task is suspended and its active jobs discarded.
- If $m \notin \mathcal{L}_i$ and $m' \in \mathcal{L}_i$, the task is activated and may immediately start releasing jobs.
- If $m, m' \in \mathcal{L}_i$, the task remains active, but its parameters are immediately changed to those at criticality level $m'$. This also affects any active (carry-over) job of the task, which will have its absolute deadline and execution-time budget immediately updated. If $C_i(m) > C_i(m')$ and a carry-over job has already executed for at least $C_i(m')$ before the mode switch, the job’s execution-time budget in $m'$ is considered to be spent, but not exceeded; the job must therefore be stopped or trigger another mode switch. The first new job of $\tau_i$ in $m'$ can be released $T_i(m')$ time units after the task’s last job release in previous modes.

The system may start in any criticality mode $m \in V(G)$ that has no incoming edges in $E(G)$. The set of such vertices is denoted $\text{roots}(G)$. We expect most systems to have only one possible start mode. Also, let

$$\text{pred}(m) \overset{\text{def}}{=} \{ m' \mid (m', m) \in E(G) \} \quad \text{and} \quad \text{succ}(m) \overset{\text{def}}{=} \{ m' \mid (m, m') \in E(G) \}.$$ 

Another aspect of the semantics that must be revisited is when a system should switch between criticality modes. In Section 2 we stated, in common
with previous work on mixed-criticality scheduling, that a system switches to a higher criticality mode if some job has executed for its entire execution-time budget without signaling completion, i.e., if a job behaves in a manner that is not valid in the current criticality mode. Similarly, the generalized task model requires that a system must switch to a new criticality mode if any job or task fails to behave in a valid manner for the current mode. However, while the system must switch modes in such a situation, it is also allowed to switch to a new criticality mode at any other point in time, for whatever reason the system designer deems relevant, e.g., because of hardware malfunctions or changes in the system’s environment. In fact, the analysis presented so far for the standard mixed-criticality model is already safe in the face of such arbitrary mode switches, because nowhere does it assume that some job has depleted its execution-time budget at the time point where a mode switch occurs. There are no restrictions on how long the system stays in any particular criticality mode before some event triggers a mode switch; it may stay there indefinitely or move on to a new mode immediately.

For the remainder of this paper we make one simplifying assumption to the above model: If \( m, m' \in \mathcal{L}_i \) and \((m, m') \in E(G)\), then it makes no sense to have \( D_i(m) > D_i(m') \) because any job of \( \tau_i \) has to be finished within \( D_i(m') \) of its release also in mode \( m \), or it would have already missed its deadline in case the system switches to \( m' \) after that time. We therefore assume that \( D_i(m) \leq D_i(m') \) if \( m, m' \in \mathcal{L}_i \) and \((m, m') \in E(G)\). We refer to this as the non-decreasing deadline invariant.

### 6 Extending the Schedulability Analysis to the Generalized Task Model

The schedulability analysis in Section 3 must be adapted to the generalized task model. This is mainly done by generalizing the demand bound functions presented previously.

Let \( \text{dbf}_{m,m'}(\tau_i, \ell) \) denote a demand bound function of task \( \tau_i \) for a time-interval length \( \ell \), when the system is currently in criticality mode \( m' \) and was in criticality mode \( m \) before that. If there was no previous mode to \( m' \), i.e., if \( m' \in \text{roots}(G) \), the demand bound function is instead denoted \( \text{dbf}_{\perp,m'}(\tau_i, \ell) \).

To avoid naming collisions, we assume that no criticality mode is ever denoted with the symbol \( \perp \). The reason demand bound functions must be formulated with both a current and a previous criticality mode in mind is that we must know if and how a task can have carry-over jobs from the previous mode.

\[^4\]If the behavior is not valid in any criticality mode that the system can switch to either, the system is considered erroneous.

\[^5\]In practice, a preprocessing step can just set \( D_i(m) \leftarrow D_i(m') \) if \( D_i(m) > D_i(m') \) in such a case. The purpose of the non-decreasing deadline invariant is not to restrict the expressiveness of the task model, but to increase the conciseness of the schedulability analysis by removing cases that can trivially be seen to not lead to schedulability.
Note that a demand bound function \( dbf_{m,m'}(\tau_i, \ell) \), as defined above, must always provide a safe upper bound on the demand of \( \tau_i \) in \( m' \) when reached from \( m \), for any possible time interval of length \( \ell \). In particular, it must provide a safe bound no matter how the system earlier reached mode \( m \), and no matter how long the system stayed in \( m \) before switching to \( m' \). A \( dbf_{m,m'}(\tau_i, \ell) \) is therefore an abstraction of all concrete system traces where \( m' \) is reached from \( m \).

With the above notation for demand bound functions, Proposition 3.1 has a natural extension:

**Proposition 6.1.** A (generalized) mixed-criticality task set \( \tau \) with criticality-mode structure \( G \) is schedulable by EDF if the following holds for all \( m \in V(G) \):

\[
\text{Condition } S(m): \quad \forall m' \in P(m) : \forall \ell \geq 0 : \sum_{\tau_i \in \tau} dbf_{m',m}(\tau_i, \ell) \leq sbf_m(\ell),
\]

where

\[
P(m) = \begin{cases} 
\text{pred}(m), & \text{if } \text{pred}(m) \neq \emptyset, \\
\{\bot\}, & \text{otherwise},
\end{cases}
\]

and where the platform’s supply in criticality mode \( m \) is characterized by supply bound function \( sbf_m \).

For each criticality mode \( m \in V(G) \), Condition \( S(m) \) captures the schedulability of the system in that mode. Condition \( S(m) \) generalizes Conditions \( S_{LO} \) and \( S_{HI} \) from Proposition 3.1, and expresses that the system’s execution demand never exceeds the available supply in mode \( m \). If Condition \( S(m) \) holds, then \( m \) is schedulable when reached from all of \( m' \)’s possible predecessor modes in \( G \), or as a start mode if \( m \) has no predecessors. If \( S(m) \) holds for all \( m \in V(G) \), then all modes of the system are schedulable, no matter how they are reached, as is stated by Proposition 6.1.

When formulating the demand bound functions later in this section, we will make two assumptions:

1. Each supply bound function \( sbf_m \) is of at most unit speed if \( \text{succ}(m) \neq \emptyset \), similarly to what was assumed of \( sbf_{LO} \) in Section 3. This is, again, simply a matter of scaling the parameters.

2. When formulating \( dbf_{m,m'} \), where \( m \neq \bot \), we assume that \( m \) is schedulable by EDF. This is analogous to what was done in Section 3, where the demand in \( HI \) was bounded under the assumption that \( LO \) is schedulable.

The second assumption clearly restricts the correctness of the demand bound functions to certain cases.\(^6\) However, our purpose with the demand bound

\(^6\)This is not an issue that can be avoided. Some knowledge about a system’s behavior prior to entering a new criticality mode must be assumed in order to provide any usable bound at all.
functions is to use them with Proposition 6.1 to show that a system is schedulable, and restricting them in this way does not invalidate their use in Proposition 6.1. In other words, if $S(m)$ holds for all $m \in V(G)$, the system is schedulable despite the above assumptions made for the demand bound functions. To see this, consider a sequence $\mathcal{T}$ that is any topological ordering $\langle m_0, m_1, \ldots \rangle$ of $G$. The assumptions made when bounding the demand in a mode $m$ are that all of $m$’s predecessors are schedulable. For $m_0$, the first mode in $\mathcal{T}$, this is trivially true (as $m_0$ can have no predecessors), and we can conclude that the bounds are valid and therefore that $m_0$ is schedulable. If $m_1$, the next mode in $\mathcal{T}$, has any predecessor in $G$, it must be $m_0$. We have already concluded that $m_0$ is schedulable, so any assumptions about the schedulability of $m_1$’s predecessors are also true and $m_1$ is also schedulable. The same reasoning can then be applied, in order, to the remaining modes $\mathcal{T}$ to see that they are all schedulable.

### 6.1 Formulating the Generalized Demand Bound Functions

There can be no carry-over jobs in any of the criticality modes in $\text{roots}(G)$ because there are no previous modes from which they can be carried over. The demand bound function $\text{dbf}_{\perp, m}$ therefore does not take carry-over jobs into account, and can be based on the standard demand bound function for sporadic tasks (Baruah et al., 1990), just like $\text{dbf}_{\text{LO}}$ in (1). The only difference is that $\text{dbf}_{\perp, m}$ is defined to be equal to 0 for tasks that are not active in $m$.

$$\text{dbf}_{\perp, m}(\tau_i, \ell) \overset{\text{def}}{=} \begin{cases} \left\lceil \frac{\ell - D_i(m)}{T_i(m)} \right\rceil + 1 \cdot C_i(m), & \text{if } m \in \mathcal{L}_i \\ 0, & \text{otherwise} \end{cases}$$

Similarly, $\text{dbf}_{\text{HI}}$ from (4) can be used as the basis for the new $\text{dbf}_{m, m'}$, as it captures the demand in modes that can be switched to and must consider carry-over jobs for tasks that are active in both $m$ and $m'$. In the same manner as $\text{dbf}_{\text{HI}}$, the function $\text{dbf}_{m, m'}$ provides a safe upper bound on the demand in $m'$ under the assumption that $m$ is schedulable. Note that as $D_i(m) \leq T_i(m)$ for all $\tau_i \in \tau$ and $m \in \mathcal{L}_i$, there can be at most one active job per task at any time point (as long as no deadline is missed). This holds also at the time of a mode switch, and so we need to consider at most one carry-over job per demand bound function, like before with $\text{dbf}_{\text{HI}}$. Recall that $\text{dbf}_{\text{HI}}$ was built from the two functions $\text{full}_{\text{HI}}$ and $\text{done}_{\text{HI}}$, from (2) and (3), respectively. We start by generalizing these two auxiliary functions.

The challenge in extending $\text{full}_{\text{HI}}$ and $\text{done}_{\text{HI}}$ to the generalized task model lies in dealing with the fact that all of a task’s worst-case execution time, relative deadline and period may change between $m$ and $m'$. However, the actual changes needed to the functions are few. The execution-time parameter could change between criticality modes already in the standard model, although it
could only increase in the new mode, and changing relative deadlines were al-
ready introduced as a technique for scheduling and analysis. The new aspects
that must be considered are therefore only the possibilities for a task to get a
decreased worst-case execution time, and a decreased or increased period. It
turns out that changing the period parameter when switching to a new mode
does not complicate the demand bound functions at all, as it does not affect
the way we calculate demand for carry-over jobs or for jobs released in the
new mode. Fig. 8 illustrates this. The only new thing that needs to be handled
is then the case where the execution-time parameter decreases.

Switch from mode \( m \) to \( m' \)

\[
D_i(m') - D_i(m)
\]

\[
T_i(m)
\]

\[
T_i(m')
\]

\[
T_i(m')
\]

\[
\cdots \rightarrow \text{Time}
\]

Figure 8. The carry-over job is unaffected by the fact that the period of \( \tau_i \)
was changed at the switch from \( m \) to \( m' \). The minimum (remaining)
scheduling window of a carry-
over job is still the difference between the relative deadlines in the new and old mode,
just as before with the standard model. As before, the remaining execution time budget
for the job in the old mode \( m \) can be no larger the length of the interval between switch
and deadline in \( m \) (the shaded interval).

The function \( \text{full}_{m,m'}(\tau_i, \ell) \) captures the demand of \( \tau_i \) in the new mode
\( m' \) without considering that carry-over jobs can be partly executed in \( m \), i.e.,
it counts a full \( C_i(m') \) also for carry-over jobs. For this it does not matter
what the execution-time parameter was in \( m \), so the function only needs to be
updated to use the relevant parameters of the generalized model:

\[
\text{full}_{m,m'}(\tau_i, \ell) \overset{\text{def}}{=} \left\lfloor \left( \frac{\ell - (D_i(m') - D_i(m))}{T_i(m')} \right) + 1 \right\rfloor \cdot C_i(m') \tag{6}
\]

The function \( \text{done}_{m,m'}(\tau_i, \ell) \) quantifies the amount of work that must have
been finished before switching to \( m' \) for any carry-over job that is included
in \( \text{full}_{m,m'}(\tau_i, \ell) \). If \( C_i(m) > C_i(m') \), it is possible that the amount of work
already finished before \( m' \) exceeds \( C_i(m') \). In such a case the carry-over
job’s execution-time budget in \( m' \) is considered to be completely spent, but
not exceeded (as per the semantics in Section 5.1), and we can only subtract
\( C_i(m') \) from \( \text{full}_{m,m'}(\tau_i, \ell) \). This is achieved by simply bounding the value of
\( \text{done}_{m,m'}(\tau_i, \ell) \) by \( C_i(m') \) from above, otherwise it is a direct adaptation of
\( \text{done}_{m1} \).
\[
\text{done}_{m,m'}(\tau_i, \ell) \overset{\text{def}}{=} \begin{cases} 
[C_i(m) - x + D_i(m') - D_i(m)]_0^{C_i(m')}, & \text{if } D_i(m') > x \geq D_i(m') - D_i(m) \\
0, & \text{otherwise},
\end{cases}
\]  
(7)

where \( x = \ell \mod T_i(m'). \)

With these auxiliary functions generalized we can formulate \( \text{dbf}_{m,m'}. \) For a task that is not active in \( m, \) the new mode \( m' \) is equivalent to a start mode in which the task may be activated, and its demand can be captured with \( \text{dbf}_{\perp,m'}. \) For a task that is active in both modes, the demand is given by the difference of \( \text{full}_{m,m'} \) and \( \text{done}_{m,m'}. \)

\[
\text{dbf}_{m,m'}(\tau_i, \ell) \overset{\text{def}}{=} \begin{cases} 
\text{full}_{m,m'}(\tau_i, \ell) - \text{done}_{m,m'}(\tau_i, \ell), & \text{if } m, m' \in \mathcal{L}_i \\
\text{dbf}_{\perp,m'}(\tau_i, \ell), & \text{otherwise}
\end{cases}
\]  
(8)

Note that if a standard two-level mixed-criticality task set is described in the generalized syntax, we have \( \text{dbf}_{\text{LO}} = \text{dbf}_{\perp,\text{LO}} \) and \( \text{dbf}_{\text{HI}} = \text{dbf}_{\text{LO,HI}} \) as expected. Also note that the \( \text{dbf}_{m,m'} \) presented above depends only on the parameters of jobs in \( m \) and \( m', \) and not on any execution history except for the assumption of schedulability in \( m. \) It is indifferent to the origin mode of carry-over jobs, i.e., if they are released in \( m \) or in an earlier mode and carried over via \( m \) to \( m'. \)

7 Tuning Parameters for the Generalized Task Model

In Section 4 we showed how it is possible to shape the demand of a task set by tuning the relative deadlines of tasks. For the standard task model, the demand of carry-over jobs in \( \text{HI} \) was reduced by decreasing the corresponding relative deadlines in \( \text{LO}. \) A similar approach can be used to shape the demand of a generalized task system \( (\tau, G) \): If \((m, m') \in E(G) \) and \( m, m' \in \mathcal{L}_i, \) the demand of carry-over jobs of \( \tau_i \) in mode \( m', \) when reached from \( m, \) can be reduced by decreasing \( D_i(m). \)

Lemma 4.1 provided insights about the effects of tuning relative deadlines for the standard task model. It can be extended to the generalized model:

Lemma 7.1 (Generalized shifting). If tasks \( \tau_i \) and \( \tau_j \) are active in mode \( m \) and are identical (i.e., have equal parameters in all modes and \( \mathcal{L}_i = \mathcal{L}_j \)), with the exception that \( D_i(m) = D_j(m) - \delta \) for \( \delta \in \mathbb{Z}, \) then

\[
\forall m' \in P(m, \tau_i) : \quad \text{dbf}_{m',m}(\tau_i, \ell) = \text{dbf}_{m',m}(\tau_j, \ell + \delta),
\]

\[
\forall m' \in S(m, \tau_i) : \quad \text{dbf}_{m,m'}(\tau_i, \ell) = \text{dbf}_{m,m'}(\tau_j, \ell - \delta),
\]

where \( P(m, \tau_i) = (\text{pred}(m) \cap \mathcal{L}_i) \cup \{ \perp \} \) and \( S(m, \tau_i) = \text{succ}(m) \cap \mathcal{L}_i. \)

\[\square\]
A proof of the above lemma is similar to that of Lemma 4.1, and is therefore omitted. Note that the correctness of our schedulability analysis or demand shaping does not depend on the correctness of this lemma, it serves only as an illustration of the effects of tuning relative deadlines.

The process of finding suitable values for the relative deadlines is more challenging for the generalized model due to the presence of arbitrarily many criticality modes and their non-linear structure. Our approach to the tuning is to first adapt Algorithm 1, which tunes the relative deadlines in one mode (LO), into the slightly more general Algorithm 3 (TuneMode(m)), which tunes the deadlines of any mode m. TuneMode(m) tries to lower the deadlines in m until all modes in succ(m) are schedulable when reached from m. Algorithm 2 (TuneSystem(τ, G)) applies Algorithm 3 on all the modes of the criticality-mode structure G, starting with the terminal nodes and proceeding in a reverse topological order until it has successfully tuned all modes or failed in tuning some mode.

**Algorithm 2: TuneSystem(τ, G)**

```
1 begin
2 for m ∈ V(G), sorted in reverse topological order do
3     if TuneMode(m) = FAILURE then return FAILURE
4     return SUCCESS
```

TuneMode(m) decreases the relative deadlines of tasks in m to reduce the demand of those tasks in m’s successor modes. Unless m ∈ roots(G), it does this without considering of the schedulability in m itself, hoping that m can later be made schedulable by decreasing deadlines in its predecessor modes. TuneMode(m) will undo changes to deadlines that would make m unschedulable only if m ∈ roots(G).

Note that TuneMode(m) will only return the value SUCCESS if it has found values for the relative deadlines in m with which all m′ ∈ succ(m) are schedulable when reached from m (assuming that m itself is ultimately made schedulable). In addition, if m ∈ roots(G), it will only return SUCCESS if m is also schedulable as a start mode. It is enough to check all integer ℓ ≤ ℓ_max to determine schedulability by the reasoning given in Section 4.1. A value for the bound ℓ_max can be computed as max{ℓ_max m′ | m′ ∈ succ(m) ∪ {m}}, where ℓ_max m′ is easily defined similarly to ℓ_max LO and ℓ_max HI from Section 4.1 when applicable, or as the hyperperiod in m. If ℓ_max is pseudo-polynomial, then so is the complexity of Algorithm 3. Algorithm 2 makes |V(G)| calls to Algorithm 3, and therefore scales linearly in the number of criticality modes.

Lines 6-9 and 28 in Algorithm 3 ensure that C_i(m) ≤ D_i(m) and the non-decreasing deadline invariant hold. In contrast to Algorithm 1, which can only undo the latest deadline change if it harms the schedulability in LO,
Algorithm 3: TuneMode(m)

1 begin
2 if succ(m) = ∅ ∧ m ∉ roots(G) then return SUCCESS
3 candidates ← {i | m ∈ L_i ∧ L_i ∩ succ(m) ≠ ∅}
4 mods ← empty stack
5 ℓ_max ← upper bound for ℓ
6 for i ∈ candidates do
7     D_i(m) ← min {D_i(m') | m' ∈ (succ(m) ∩ L_i) ∪ {m}}
8     if D_i(m) = C_i(m) then candidates ← candidates \ {i}
9     if D_i(m) < C_i(m) then return FAILURE
10 repeat
11     changed ← false
12     for ℓ = 0, 1, ..., ℓ_max do
13         if m ∈ roots(G) then
14             if ∑ τ_i ∈ τ dbf_m,m(τ_i, ℓ) > sbf_m(ℓ) then
15                 if mods is empty then return FAILURE
16                 i ← mods.pop()
17                 D_i(m) ← D_i(m) + 1
18                 candidates ← candidates \ {i}
19                 changed ← true
20                 break
21     for m' ∈ succ(m) do
22         if ∑ τ_i ∈ τ dbf_m,m'(τ_i, ℓ) > sbf_m'(ℓ) then
23             c ← candidates ∩ {i | m' ∈ L_i}
24             if c = ∅ then return FAILURE
25             i ← arg max_j∈c(dbf_m,m'(τ_j, ℓ) - dbf_m,m'(τ_j, ℓ - 1))
26             D_i(m) ← D_i(m) - 1
27             mods.push(i)
28             if D_i(m) = C_i(m) then
29                 candidates ← candidates \ {i}
30                 changed ← true
31                 break
32 until ¬changed
33 return SUCCESS

Algorithm 3 stores all deadline changes in a stack (mods) and can undo all of them in an effort to restore schedulability of a start mode. This does not affect the scalability much, as each deadline parameter D_i(m) can only be changed 2 · (D − C_i(m)) times at most, where D = D_i(m) at the start of Algorithm 3.
It is of course possible to define Algorithm 3 without the stack (or Algorithm 1 with it), but we think that it is motivated to be able to undo more changes in Algorithm 3 as there can be several successor modes incurring them.

Because Algorithm 2 traverses the DAG in a reverse topological order, we know that all successor modes of modes tuned so far will be schedulable as long as we can successfully tune the remaining modes. Example 7.1 illustrates this process.

**Example 7.1.** Let task set $\tau$ have criticality-mode structure $G$. Let $V(G) = \{m_0, m_1, m_2, m_3\}$ and $E(G) = \{(m_0, m_1), (m_0, m_2), (m_1, m_3), (m_2, m_3)\}$, resulting in a shape similar to the one in Example 5.1. A reverse topological ordering of the criticality modes is $\langle m_3, m_1, m_2, m_0 \rangle$. $\text{TuneSystem}(\tau, G)$ would first call $\text{TuneMode}(m_3)$, which does nothing as $\text{succ}(m_3) = \emptyset$, and then continue with $m_1, m_2, m_0$ in order. As shown in Fig. 9, $\text{TuneMode}(m_1)$ and $\text{TuneMode}(m_2)$ will decrease some relative deadlines in $m_1$ and $m_2$, as needed to make $m_3$ schedulable when reached from either of these modes. $\text{TuneMode}(m_0)$ will decrease deadlines in $m_0$ to make both $m_1$ and $m_2$ schedulable (using the lowered deadlines previously set in $\text{TuneMode}(m_1)$ and $\text{TuneMode}(m_2)$). Because $m_0 \in \text{roots}(G)$, $\text{TuneMode}(m_0)$ will also make sure that $m_0$ is schedulable as a start mode, or return FAILURE. □

![Figure 9](image-url) The order in which modes are tuned by $\text{TuneSystem}(\tau, G)$, excluding the trivial $\text{TuneMode}(m_3)$

### 8 Experimental Evaluation

In this section we evaluate the effectiveness of characterizing mixed-criticality task sets using the demand bound functions formulated earlier. In order to make the evaluation meaningful, we compare to previous approaches to mixed-criticality scheduling from the literature and study the acceptance ratios of their corresponding schedulability tests. The previous work only supports dedicated platforms, and therefore we use that setting for the experiments. Most of the previous work assumes the standard two-level mixed-criticality sporadic task model described in Section 2, so we start by considering such task sets.
8.1 Evaluation Using the Standard Two-Level Task Model

We compare the following approaches:

**GreedyTuning:** The test in Proposition 3.1 using the demand bound functions in Equations (1) and (4). Relative deadlines are tuned using Algorithm 1.

**OCBP-prio:** The test for OCBP-based scheduling by Guan et al. (2011), which is based on whether a priority ordering can be found for all jobs in a busy period. This test has been shown to dominate the test for OCBP-based scheduling by Li and Baruah (2010), and is therefore the only test for OCBP included.

**EDF-VD:** The test for the EDF-VD scheduling algorithm by Baruah et al. (2012).

**AMC-max:** A test based on the most powerful response-time calculation for fixed-priority scheduling by Baruah et al. (2011c), called AMC-max. Priorities are assigned using Audsley’s algorithm, as suggested by Baruah et al. (2011c).

**Vestal:** A test based on the response-time calculation for fixed-priority scheduling by Vestal (2007) combined with Audsley’s algorithm. Because we assume that low-criticality tasks are discarded in high-criticality mode, the budgets of low-criticality task’s execution times are implicitly enforced. This is therefore equivalent to the algorithm SMC by Baruah et al. (2011c).

**Naive:** A test based on simply flattening the mixed-criticality sporadic task set into a normal sporadic task set using resource reservation, and seeing whether the constructed task set is schedulable. In the case of implicit deadline tasks this is done by simply checking if the utilization of the constructed task set is at most 1. If deadlines are not implicit, the exact test by Baruah et al. (1990) is used instead. Each mixed-criticality task $\tau_i \in \tau$ is mapped to a normal sporadic task with worst-case execution time $C_i(L_i)$, deadline $D_i$ and period $T_i$. This simple test is included as a baseline for the more sophisticated approaches.

**Necessary:** This is an over-approximation of task set feasibility. A task set $\tau$ passes this test if both sporadic task sets $\{(C_i(LO), D_i, T_i) \mid \tau_i \in \tau\}$ and $\{(C_i(H1), D_i, T_i) \mid \tau_i \in \text{Hi}(\tau)\}$ are schedulable according to the exact analysis by Baruah et al. (1990). This test is helpful in providing a bound on how much improvement upon existing solutions we can ever hope to achieve. It is not clear, of course, how close the upper bound provided by this test is to true feasibility. It follows from the work of Baruah et al. (2011a) that an exact feasibility test, should one be developed, is NP-

---

7A similar evaluation performed in a preliminary version of this paper (Ekberg and Yi, 2012) used the original test for EDF-VD (Baruah et al., 2011b). Here we use the improved test (Baruah et al., 2012).
hard in the strong sense. We only use the necessary test if deadlines are not implicit, as all implicit-deadline task sets are guaranteed to pass it.

8.1.1 Task Set Generation

A random task set is generated by starting with an empty task set \( \tau = \emptyset \), which random tasks are successively added to. The generation of a random task is controlled by five parameters: the probability \( P_{HI} \) of being of high-criticality; the maximum ratio \( R_C \) between high- and low-criticality execution time; the maximum low-criticality execution time \( C_{LO}^{\text{max}} \); the maximum period \( T_{max} \); and the ratio \( R_D \) giving the range of possible relative deadline values. Let \( \mathcal{U}\{\cdots\} \) denote the discrete uniform distribution over some finite set \{\cdots\}.

Each new task \( \tau_i \) is then generated as follows:

- \( L_i = HI \) with probability \( P_{HI} \), otherwise \( L_i = LO \).
- \( C_i(LO) \) is drawn from \( \mathcal{U}\{1, \ldots, C_{LO}^{\text{max}}\} \).
- \( C_i(HI) \) is drawn from \( \mathcal{U}\{C_i(LO), \ldots, R_C \cdot C_i(LO)\} \) if \( L_i = HI \), otherwise \( C_i(HI) = C_i(LO) \).
- \( T_i \) is drawn from \( \mathcal{U}\{C_i(L_i), \ldots, T_{max}\} \).
- \( D_i \) is drawn from \( \mathcal{U}\{D_{min}, \ldots, T_i\} \), where \( D_{min} \) is calculated based on \( R_D \) as

\[
D_{min} = C_i(L_i) + R_D \cdot (T_i - C_i(L_i))
\]

We define the average utilization \( U_{avg}(\tau) \) of a mixed-criticality task set \( \tau \) as

\[
U_{avg}(\tau) \overset{\Delta}{=} \frac{U_{LO}(\tau) + U_{HI}(\tau)}{2}.
\]

Each task set is generated with a target average utilization \( U^* \) in mind. Due to the difficulty of getting an exact utilization with random integer parameter tasks, we allow the task set’s average utilization to fall within the small interval between \( U_{min}^* \overset{\Delta}{=} U^* - 0.005 \) and \( U_{max}^* \overset{\Delta}{=} U^* + 0.005 \).

As long as \( U_{avg}(\tau) < U_{min}^* \), we generate more tasks and add them to \( \tau \). If a task is added such that \( U_{avg}(\tau) > U_{max}^* \), we discard the whole task set and start with a new empty task set. If a task is added such that \( U_{min}^* \leq U_{avg}(\tau) \leq U_{max}^* \), the task set is finished, unless all tasks in \( \tau \) have the same criticality level or \( U_{LO}(\tau), U_{HI}(\tau) > 0.99 \), in which case the task set is instead discarded.\( ^8 \)

8.1.2 Results

Fig. 10 shows the acceptance ratio (fraction of schedulable task sets) as a function of (target) average utilization for task sets generated with \( P_{HI} = 0.5 \),

\(^8 The reason we use a maximum utilization of 0.99 instead of 1 is to be able to use a pseudo-polynomial upper bound \( \ell_{max} \) for Algorithm 1 instead of the hyperperiod, as described in Section 4.1. This speeds up the run-time of the experiments significantly, which is needed as several million task sets are analyzed in total.
Next we study the effects of varying the parameters $P_{hi}$, $R_C$ and $R_D$. We plot the weighted acceptance ratio (called the weighted schedulability measure by Bastoni et al. (2010)) as a function of the varied parameter. If $A(U)$ is the acceptance ratio for (target) average utilization $U$, then the weighted acceptance ratio of a set of target utilizations $\Upsilon$ is

$$A(\Upsilon) = \frac{\sum_{U \in \Upsilon} U \cdot A(U)}{\sum_{U \in \Upsilon} U}.$$ 

Using the weighted acceptance ratio we can reduce the number of dimensions in the plots by one. Note that this measure gives more importance to the acceptance ratios for larger utilization values, as these are the cases we are generally interested in.

In Fig. 11, 12 and 13 we have plotted the weighted acceptance ratio as a function of $P_{hi}$, $R_C$ and $R_D$, respectively. The set of average utilization values is $\Upsilon = \{(1/30) \cdot (x + 1/2) \mid x \in \{0, \ldots, 29\}\}$, i.e., the same 30 values that is used in Fig. 10. Except for the varied parameter, the parameters are also the same as for Fig. 10 to allow easy comparisons. Each data point is based on 30,000 random task sets (1000 per target utilization in $\Upsilon$). Note that in Fig. 13 we have omitted EDF-VD, as it only supports implicit deadline tasks.
We have instead added the Necessary test to provide an over-approximation on feasibility.

![Graph showing acceptance ratios for different approaches]

Figure 11. $P_{HI}$ varying, $R_C = 4$, $C_{LO}^{\text{max}} = 10$, $T^{\text{max}} = 200$ and $R_D = 1$

8.1.3 Discussion

Evidently, for the standard task model there is often a large gap between the acceptance ratios of the proposed approach in this paper and those of previous approaches. Moreover, this gap remains when varying the fraction of high-criticality tasks ($P_{HI}$) or the ratio between low- and high-criticality worst-case execution times ($R_C$). However, the differences between acceptance ratios decrease when relative deadlines become shorter ($R_D$ gets smaller). When relative deadlines can range from being no larger than worst-case execution times ($R_D = 0$) all compared approaches have quite similar (weighted) acceptance ratios. The results can of course differ from those presented if we vary other parameters or the task set generation procedure, but the typical gap between acceptance ratios is large enough that we think it is safe to say that the proposed approach in this paper marks a significant improvement in the scheduling of standard mixed-criticality sporadic task sets with two criticality levels.

Among previous approaches studied, OCBP-prio (Guan et al., 2011), AMC-max (Baruah et al., 2011c) and EDF-VD (Baruah et al., 2012) seem to perform best. Of these, AMC-max and EDF-VD are probably the best choices in prac-
Figure 12. $P_{HI} = 0.5$, $R_C$ varying, $C_{LO}^{\text{max}} = 10$, $T^{\text{max}} = 200$ and $R_D = 1$

Figure 13. $P_{HI} = 0.5$, $R_C = 4$, $C_{LO}^{\text{max}} = 10$, $T^{\text{max}} = 200$ and $R_D$ varying
tice as they have a significantly lower run-time overhead than OCBP-based scheduling. The run-time overhead of our approach is also low because it is basically just plain EDF (potentially with a change of deadlines at single point in time), the same as EDF-VD.

The weighted acceptance ratios of all approaches remain relatively steady when varying $P_{HI}$ and $R_C$. The reasons for the slow trends that can be seen in Fig. 11 and 12 remain mostly unclear to us. An exception is $R_C = 1$, with which worst-case execution times do not differ between low- and high-criticality modes. Such a task set is actually equivalent to a standard sporadic task set, which is why the baseline (Naive) and EDF-VD approaches have 100% acceptance ratios (both reduce in this case to checking whether the utilization is at most 1). When $R_D$ decreases, the acceptance ratios of all compared approaches decrease as well, as seen in Fig. 13. This is to be expected as the timing constraints become tougher with smaller deadlines. The approach proposed in this paper sees a more marked decrease in acceptance ratio than the others, we think this is explained by its heavy dependency on being able to shift demand between criticality modes by tuning the relative deadline parameters. When these become smaller, less demand can be shifted. However, it should be noted that the upper bound on feasibility provided by the necessary test sees a decrease in acceptance ratio that is almost as sharp; it may be that the room for improvement in this regard is fairly limited.

8.2 Evaluation Using the Pessimistic Frequency Specification Model

Baruah (2012) introduced a system model that is similar to the standard mixed-criticality model, but where the periods of tasks, instead of their execution-time budgets, can change between the criticality modes LO and HI. Any such task set can be expressed as an instance $(\tau, G)$ of the generalized mixed-criticality model described in Section 5, where $V(G) = \{LO, HI\}$, $E(G) = \{(LO, HI)\}$ and for each task $\tau_i \in \tau$:

- $L_i = \{LO\}$ or $L_i = \{LO, HI\}$.
- If HI $\in L_i$, then
  - $C_i(HI) = C_i(LO)$,
  - $T_i(HI) \leq T_i(LO)$,
  - $D_i(LO) = D_i(HI) = T_i(HI)$.
- Otherwise, $D_i(LO) = T_i(LO)$.

The following approaches are compared:

**TuneSystem:** The test in Proposition 6.1 using the demand bound functions in Equations (5) and (8). Relative deadlines are tuned using Algorithm 2.
Baruah: The test by Baruah (2012) based on finding a smallest parameter $x \in (0, 1)$ satisfying certain requirements, and then abstracting the workload in each criticality mode with normal sporadic tasks, specially constructed using $x$. It is suggested by Baruah (2012) that a smallest $x$ satisfying the requirements can be found using bisection search. We did this and made sure that the precision was at least 0.001. We found that further increasing the precision did not noticeably affect the results.

Naive: A baseline test based on flattening the mixed-criticality sporadic task set into a normal sporadic task set using resource reservation, and then checking whether the utilization of the constructed task set is at most 1. Each mixed-criticality task $\tau_i \in \tau$ is mapped to a normal implicit-deadline sporadic task with worst-case execution time $C_i(LO)$ and period $T_i(HI)$ if $HI \in \mathcal{L}_i$ and period $T_i(LO)$ otherwise.

8.2.1 Task Set Generation

The generation of random task sets is controlled by four parameters: the probability $P_{HI}$ of high criticality; the maximum execution time $C_{max}$; the maximum period $T_{max}$; and the minimum ratio $R_T$ between the periods in high-and low-criticality mode. Each random task is generated as follows.

- $\mathcal{L}_i = \{LO, HI\}$ with probability $P_{HI}$, otherwise $\mathcal{L}_i = \{LO\}$.
- $C_i(LO)$ is drawn from $\mathcal{U}\{1, \ldots, C_{max}\}$.
- $T_i(LO)$ is drawn from $\mathcal{U}\{C_i(LO), \ldots, T_{max}\}$.
- If $HI \in \mathcal{L}_i$, then
  - $C_i(HI) = C_i(LO)$,
  - $T_i(HI)$ is drawn from $\mathcal{U}\{T_{min}, \ldots, T_i(LO)\}$, where we calculate $T_{min}$ based on $R_T$ as $T_{min} = \max(C_i(HI), \lceil R_T \cdot T_i(LO) \rceil)$,
  - $D_i(LO) = D_i(HI) = T_i(HI)$.
- If $HI \not\in \mathcal{L}_i$, then $D_i(LO) = T_i(LO)$.

Average utilization for the generalized task model is defined as

$$U_{avg}(\tau) \overset{def}{=} \frac{\sum_{m \in V(G)} U_m(\tau)}{|V(G)|}.$$  

The task sets are then generated with a target average utilization $U^*$ in mind, in the same manner as in Section 8.1.1. Only task sets $\tau$ with $U^* - 0.005 \leq U_{avg}(\tau) \leq U^* + 0.005$ and with $\forall m \in V(G): U_m(\tau) \leq 0.99$ are kept.

8.2.2 Results

Fig. 14 shows the acceptance ratios as a function of target average utilization for task sets generated with $P_{HI} = 0.5$, $R_T = 0.25$, $C_{max} = 10$ and $T_{max} =$
200. These values were chosen to mirror those used for Fig. 10, with the difference that here the periods in H1 may be up to four times shorter, instead of execution times being up to four times longer. Each data point is based on 10,000 random task sets.

Next we study the effects of varying the value of $R_T$. Fig. 15 shows the weighted acceptance ratio as a function of $R_T$. The set of target utilizations is $\Omega = \{(1/30) \cdot (x + 1/2) \mid x \in \{0, \ldots, 29\}\}$. Each data point is based on 30,000 task sets.

8.2.3 Discussion

From Figs. 14 and 15 it seems that the demand bound functions formulated in this paper, combined with the heuristic parameter tuning, provides a scheduling approach that mostly performs better than the currently known alternatives. However, the acceptance ratio curve of our approach is generally lower for this task model than for the standard task model, with experiment settings as similar as possible (e.g., compare Figs. 14 and 10). We think that this is mostly due to the generally lower periods (and relative deadlines) of critical tasks in this model, which leave less room for parameter tuning, limiting the ability to shift demand from one criticality mode to another.

As seen in Fig. 15, all compared scheduling approaches improve with larger values for $R_T$. The naive approach gains rapidly as $R_T$ approaches 1, reaching
near perfect acceptance ratio with $R_T = 0.95$. This is similar to the effect seen in Fig. 12, and is explained by the fact that these task sets are very close to being equivalent to normal sporadic tasks sets, and so the pessimism introduced by flattening them becomes relatively insignificant.

8.3 Evaluation Using Several Linearly Ordered Criticality Modes

Some of the scheduling approaches evaluated in 8.1 support an extension of that system model with $K \geq 2$ linearly ordered criticality modes. Any such task set can be expressed as an instance $(\tau, G)$ of the generalized system model in Section 5, where

- $V(G) = \{m_1, \ldots, m_K\}$,
- $E(G) = \{(m_1, m_2), \ldots, (m_{K-1}, m_K)\}$.
- For each task $\tau_i \in \tau$,
  - $L_i = \{m_1, \ldots, m_{L_i}\}$, for some $1 \leq L_i \leq K$,
  - $C_i(m_1) \leq \cdots \leq C_i(m_{L_i})$,
  - $D_i(m_1) = \cdots = D_i(m_{L_i})$,
  - $T_i(m_1) = \cdots = T_i(m_{L_i})$.

The approaches we compare are:

**TuneSystem**: The test in Proposition 6.1 using the demand bound functions in Equations (5) and (8). Relative deadlines are tuned using Algorithm 2.
OCBP-prio: The same test as we used for OCBP-prio in Section 8.1. It supports this setting as well.

Vestal: The same test as in Section 8.1 can also be used here.

Naive: A baseline test based on flattening the mixed-criticality sporadic task set into a normal sporadic task set using resource reservation, and checking whether the constructed task set is schedulable. Each mixed-criticality task \( \tau_i \in \tau \) is mapped to a normal sporadic task with worst-case execution time \( C_i(m_{L_i}) \), deadline \( D_i(m_1) \) and period \( T_i(m_1) \).

### 8.3.1 Task Set Generation

Here the task set generation is controlled by four parameters: the number of criticality modes \( K \); the maximum execution time \( C_{m_1}^{\text{max}} \) in the lowest criticality mode; the maximum period \( T_{\text{max}} \); and the maximum ratio \( R_c \) between the execution time parameter of two consecutive criticality modes. Each task \( \tau_i \) is generated as follows:

- \( L_i = \{m_1, \ldots, m_{L_i}\} \) for an \( L_i \) drawn from \( U\{1, \ldots, K\} \),
- \( C_i(m_1) \) is drawn from \( U\{1, \ldots, C_{m_1}^{\text{max}}\} \),
- \( C_i(m_j) \) is drawn from \( U\{C_i(m_{j-1}), \ldots, R_c \cdot C_i(m_{j-1})\} \) for every \( j \), such that \( 1 < j \leq L_i \),
- \( D_i(m_1) = \cdots = D_i(m_{L_i}) = T_i(m_1) = \cdots = T_i(m_{L_i}) \) are drawn from \( U\{C_i(m_{L_i}), \ldots, T_{\text{max}}\} \).

Note that we generate tasks with implicit deadlines. This is to enable easier comparisons with previous experiments, which are mostly with implicit deadlines as well. Task sets are generated for a target average utilization in exactly the same way as in Section 8.2.1.

### 8.3.2 Results

Fig. 16 shows acceptance ratio as a function of average utilization for task sets generated with \( K = 3 \), \( C_{m_1}^{\text{max}} = 10 \), \( T_{\text{max}} = 200 \) and \( R_c = 2 \). There are 10,000 task sets per data point. Note that there are only 28 different target utilizations used, instead of the 30 used in previous experiments. We have skipped the first and the last point because it is difficult to generate random task sets fulfilling the criteria and having a very low (or very high) average utilization. If we extrapolate the acceptance ratio curves in Fig. 16, it would not seem as if the two skipped points are interesting.

The effects of having more criticality levels are shown in Fig. 17, where we vary the value of \( K \). We have not included OCBP-prio in Fig. 17 because its schedulability test becomes prohibitively expensive for larger values of \( K \) combined with high average utilization. The set of target utilizations used for
Figure 16. $R_c = 2$, $C_{m_1}^{\text{max}} = 10$, $T_{\text{max}} = 200$ and $K = 3$

the weighted acceptance ratio is $\mathcal{U} = \{(1/30) \cdot (x + 1/2) \mid x \in \{1, \ldots, 28\}\}$. Each data point is based on 30,000 task sets.

8.3.3 Discussion

As seen in Fig. 16, the scheduling approach taken in this paper performs better than alternative approaches also for standard mixed-criticality task sets extended to three criticality levels. The difference in performance, however, is much smaller than for the case with two levels (e.g., see Fig. 10).

Increasing the number of criticality levels further leads to quickly deteriorating performance for all evaluated approaches, as shown in Fig. 17. Vestal’s fixed-priority response-time analysis deteriorates most gracefully though, and for $K > 4$ it has the best performance. This is not so surprising as the response-time analysis used for it suffers comparatively little extra pessimism for increasing $K$. In contrast, the scheduling approach proposed in this paper would require the tuning to produce a differentiation of the relative deadlines of a task $\tau_i$ between each adjacent criticality mode, such that if $\mathcal{L}_i = \{m_1, \ldots, m_{L_i}\}$, we have $D_i(m_1) \leq \cdots \leq D_i(m_{L_i})$. Clearly, when a task is active in a long “chain” of criticality modes, the room available for differentiating the relative deadlines between adjacent modes becomes smaller. It may pay off to shorten such chains, if possible, when modeling the system, i.e., to make a trade-off between modeling pessimism and analysis pessimism.
9 Conclusions

We have characterized the demand of mixed-criticality sporadic tasks using demand bound functions. They are based on the observation that upper bounds on the demand of carry-over jobs can be formulated by assuming that the previous criticality mode is schedulable. As the goal of schedulability analysis in this context is to show that all criticality modes are schedulable, such assumptions are sound as long as the base cases (i.e., the starting criticality modes) can be shown to be schedulable.

We have also generalized the standard mixed-criticality sporadic task model used in most prior work. The generalized task model allows arbitrary changes of task parameters and active tasks between criticality modes. We allow the criticality modes to be expressed using a DAG, thus enabling the specification of mixed-criticality aspects in several dimensions. We would like to have a general model to cover a broad range of mixed-criticality aspects and leave the interpretations to the system designer.

For the demand bound functions that have been formulated, we have showed that demand of a task can be shifted between any adjacent criticality modes by tuning the relative deadlines of the task in different modes. This results in a constraint satisfaction problem where the challenge is to find valid values for all relative deadlines such that the whole system becomes schedulable. The tuning of parameter values can be carried out in any manner, for exam-
ple with general-purpose constraint solvers. We have also presented heuristic algorithms designed specifically for this problem. Note that the values of relative deadlines only can decrease with respect to their original values, meaning that the temporal specification made by the system designer holds also after the parameter tuning.

Experimental evaluation and comparisons between this and prior work show that our approach is successful in practice for a wide range of situations. The results show that EDF-based scheduling in many cases significantly outperforms fixed-priority scheduling for mixed-criticality systems, mirroring the case for non-mixed-criticality systems. We think that this is important because it allows us to utilize the performance of EDF without sacrificing robustness in case of overloads. Often, EDF is quoted as being too unpredictable in case of overloads since it is practically impossible to predict which jobs will suffer the extra delays (see Buttazzo (2005) for a more comprehensive treatment of this subject). This is not the case for mixed-criticality systems. In a mixed-criticality system, the designer can specify exactly which tasks are more important in an overload situation. We believe that this is an appropriate separation of concerns: The system designer specifies what constitutes an overload situation, or other critical event, and which tasks must continue to function. The scheduler makes sure that the system behaves as specified while utilizing platform resources as efficiently as possible.

As future work we plan to address resource sharing or tasks with more complex job release patterns in the context of mixed-criticality systems. We believe that the techniques in this paper will generalize to these cases.

References


Appendix A Proof of Lemma 4.1

Before proving Lemma 4.1, we reformulate the function \( \text{done}_{\text{HI}}(\tau_i, \ell) \) from Section 3.3 as

\[
\text{done}^*(\tau_i, \ell) \overset{\text{def}}{=} \lceil C_i(\text{LO}) - ((\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \mod T_i) \rceil_0
\]

and show that it is an equivalent definition:

**Lemma A.1.**

\[
\text{done}^*(\tau_i, \ell) = \text{done}_{\text{HI}}(\tau_i, \ell)
\]

**Proof.** We split the proof into three cases.

**First case:** \( D_i(\text{HI}) > \ell \mod T_i \geq D_i(\text{HI}) - D_i(\text{LO}) \).

From \( \ell \mod T_i \geq D_i(\text{HI}) - D_i(\text{LO}) \) we know that

\[
((\ell \mod T_i) - (D_i(\text{HI}) - D_i(\text{LO}))) \mod T_i = (\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \mod T_i.
\]

With (9) we can rewrite \( \text{done}^*(\tau_i, \ell) \) as

\[
\text{done}^*(\tau_i, \ell) = \lceil C_i(\text{LO}) - ((\ell \mod T_i) - (D_i(\text{HI}) - D_i(\text{LO}))) \rceil_0
\]

\[
= \lceil C_i(\text{LO}) - (\ell \mod T_i) + D_i(\text{HI}) - D_i(\text{LO}) \rceil_0
\]

\[
= \text{done}_{\text{HI}}(\tau_i, \ell).
\]
Second case: $D_i(HI) \leq \ell \mod T_i$.

From $D_i(HI) \leq \ell \mod T_i$ the equality (9) follows again. We can rewrite done ($\tau_i, \ell$):

\[
\text{done}^*(\tau_i, \ell) = \llbracket C_i(LO) - (\ell \mod T_i) + D_i(HI) - D_i(LO) \rrbracket_0
= \llbracket (C_i(LO) - D_i(LO)) + (D_i(HI) - \ell \mod T_i) \rrbracket_0
\]

We have $C_i(LO) \leq D_i(LO)$ and $D_i(HI) \leq \ell \mod T_i$. Therefore,

\[
(C_i(LO) - D_i(LO)) + (D_i(HI) - \ell \mod T_i)) \leq 0
\]

and

\[
\text{done}^*(\tau_i, \ell) = 0 = \text{done}_H(\tau_i, \ell).
\]

Third case: $\ell \mod T_i < D_i(HI) - D_i(LO)$.

From $\ell \mod T_i < D_i(HI) - D_i(LO)$ and from $C_i(LO) \leq D_i(LO) \leq D_i(HI) \leq T_i$ we have\(^9\)

\[
\begin{align*}
(\ell - (D_i(HI) - D_i(LO))) \mod T_i
&= T_i - (D_i(HI) - D_i(LO)) + (\ell \mod T_i) \\
&\geq T_i - (D_i(HI) - D_i(LO)) \\
&\geq D_i(LO) \\
&\geq C_i(LO).
\end{align*}
\]

Therefore,

\[
C_i(LO) - ((\ell - (D_i(HI) - D_i(LO))) \mod T_i) \leq 0
\]

and

\[
\text{done}^*(\tau_i, \ell) = 0 = \text{done}_H(\tau_i, \ell).
\]

We can now prove Lemma 4.1.

\[\]

Proof of Lemma 4.1. The lemma follows from straightforward substitutions. First,

\[
\text{dbf}_L(\tau_i, \ell) = \llbracket \left( \left\lfloor \frac{\ell - D_i(LO)}{T_i} \right\rfloor + 1 \right) \cdot C_i(LO) \rrbracket_0
= \llbracket \left( \left\lfloor \frac{\ell - (D_j(LO) - \delta)}{T_j} \right\rfloor + 1 \right) \cdot C_j(LO) \rrbracket_0
= \text{dbf}_L(\tau_j, \ell + \delta).
\]

\(^9\)Note that we interpret mod as positive remainder: $a \ mod \ b = a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b.$
To show the \( dbf_{HI} \) part, we consider \( \text{full}_{HI} \) and \( \text{done}_{HI} \) separately:

\[
\text{full}_{HI}(\tau_i, \ell) = \left\lfloor \left( \frac{\ell - (D_i(\text{HI}) - D_i(\text{LO}))}{T_i} \right) + 1 \right\rfloor \cdot C_i(\text{HI}) \\
= \left\lfloor \left( \frac{\ell - (D_j(\text{HI}) - (D_j(\text{LO}) - \delta))}{T_j} \right) + 1 \right\rfloor \cdot C_j(\text{HI}) \\
= \text{full}_{HI}(\tau_j, \ell - \delta)
\]

We use Lemma A.1 for \( \text{done}_{HI}(\tau_i, \ell) = \text{done}^*(\tau_i, \ell) \):

\[
\text{done}_{HI}(\tau_i, \ell) = \text{done}^*(\tau_i, \ell) \\
= \left\lfloor C_i(\text{LO}) - (\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \mod T_i \right\rfloor \\
= \left\lfloor C_j(\text{LO}) - (\ell - (D_j(\text{HI}) - (D_j(\text{LO}) - \delta))) \mod T_j \right\rfloor \\
= \text{done}^*(\tau_j, \ell - \delta) \\
= \text{done}_{HI}(\tau_j, \ell - \delta)
\]

The \( dbf_{HI} \) part follows directly:

\[
\text{dbf}_{HI}(\tau_i, \ell) = \text{full}_{HI}(\tau_i, \ell) - \text{done}_{HI}(\tau_i, \ell) \\
= \text{full}_{HI}(\tau_j, \ell - \delta) - \text{done}_{HI}(\tau_j, \ell - \delta) \\
= \text{dbf}_{HI}(\tau_j, \ell - \delta)
\]
Schedulability Analysis of a Graph-Based Task Model for Mixed-Criticality Systems

Pontus Ekberg and Wang Yi

Abstract

We present a new graph-based real-time task model that can specify complex job arrival patterns and global state-based mode switching. The mode switching is of a mixed-criticality style, meaning that it allows immediate changes to the parameters of active jobs upon mode switches. The resulting task model generalizes previously proposed task graph models as well as mixed-criticality (sporadic) task models; the merging of these mutually incomparable modeling paradigms allows formulation of new types of tasks. A sufficient schedulability analysis for EDF on preemptive uniprocessors is developed for the proposed model.

1 Introduction

During the last seven years, a wealth of research has investigated the scheduling and analysis of mixed-criticality systems, often using a sporadic mixed-criticality task model that has become a de facto standard (e.g., Vestal, 2007; Li and Baruah, 2010; Guan et al., 2011; Baruah et al., 2011b,c; Ekberg and Yi, 2012). While this model is popular and theoretically interesting, it has been criticized for its limited applicability to many real systems (e.g., see Burns and Baruah, 2013). Some of this criticism can be traced back to the task model’s restricted notion of what should happen to each task or job upon a change of the system’s criticality mode and to its lack of an explicit mechanism for going back to previous modes.

To tackle these problems and more, we present a new task model that we call the Mode-Switching Digraph Real-Time (MS-DRT) task model. It combines complex arrival patterns of jobs with global mode switching. The tasks are represented by graphs that specify both the arrival patterns of jobs and the synchronization points (mode switches) between tasks. MS-DRT is a strict generalization of the Digraph Real-Time (DRT) task model (Stigge et al., 2011) and of the common mixed-criticality sporadic task model, as well as of some of its variations (Baruah, 2012) and generalizations (Ekberg and Yi, 2014).

Mode-switching logic is specified per state (vertex) of the task graphs, so that behaviors may differ depending on the local state of the tasks. The mode
change protocol is of a generalized mixed-criticality style, enabling immediate changes to the timing parameters of active jobs at mode changes. As opposed to the usual mixed-criticality setting, the order in which different modes may be visited in MS-DRT can take the form of an arbitrary directed graph, including cycles.

The combination of graph-based task models with state-based mode switching results in a fairly general model. Its semantics need not be interpreted as those of a mixed-criticality system. It could also find use as a timing model for other types of state-based systems with modes, such as statecharts (Harel, 1987).

In this paper we describe and prove correct a sufficient schedulability analysis for EDF for the proposed task model on preemptive uniprocessors. Because of the complexity of the task model, the analysis follows a structured approach in which each mode of the system is analyzed in relative separation by abstracting the influences from other modes. With this approach it is also possible to use other scheduling algorithms in some of the modes, without the need of updating the analysis for the modes scheduled by EDF.

The analysis procedure builds upon ideas from previously published EDF schedulability analysis methods for DRT task sets (Stigge et al., 2011) and mixed-criticality sporadic task sets (Ekberg and Yi, 2012, 2014). The proposed test has the property that it is exact for the case where there is only a single mode in the system (in this case it reduces to the test for DRT task sets from Stigge et al. (2011)) and is equal\(^1\) to the schedulability test from Ekberg and Yi (2014) in the case where the modeled system is equivalent to a mixed-criticality sporadic task system. The latter test, while not being exact (in common with all other schedulability tests for sporadic mixed-criticality systems to date), has empirically been shown to perform well (see Ekberg and Yi (2014) for details). For systems that combine features from both cases—the systems that we are mainly interested in here—it is difficult to evaluate the effectiveness of the proposed test because there are no other tests to compare with for this new model. Still, there is no source of pessimism in the test other than the ones already present in Ekberg and Yi (2014), so there is reason to believe that it performs well also for some of these other systems.

1.1 Related Work

After the seminal paper by Vestal (2007), concerning fixed-priority response-time analysis for mixed-criticality sporadic task systems, the initial research effort into mixed-criticality scheduling considered static sequences of jobs. The work by Baruah et al. (2011a) provides a good overview of such mixed-criticality job scheduling. Later on, many works have considered the scheduling and analysis of mixed-criticality sporadic task systems, e.g., Li and Baruah

\(^1\)It is equal assuming the same heuristics are applied in a preprocessing tuning phase.
EDF-based scheduling of mixed-criticality sporadic task systems was investigated by Baruah et al. (2011b) in their work on EDF-VD. With EDF-VD they introduced the idea of changing the deadline of jobs upon a switch to another criticality mode. Similar EDF-based runtime scheduling was later used by Ekberg and Yi (2012), but with an analysis based on computing demand bound functions for the mixed-criticality tasks. Demand bound functions offer a useful abstraction for use in EDF-based schedulability analysis, and have been applied to many varying task models outside of the mixed-criticality setting. For example, scheduling analyses based on demand bound functions exist for task models that offer greater expressiveness than sporadic tasks regarding job arrival patterns, such as the GMF (Baruah et al., 1999) and DRT (Stigge et al., 2011) task models. This wide applicability of demand bound functions is what allows us analyze a combination of mixed-criticality style mode switching with more general job release patterns in this paper.

Easwaran (2013) and Zhang et al. (2014) have adapted the demand bound function based analysis in Ekberg and Yi (2012) by essentially breaking the relative isolation in which modes are considered, thereby increasing both computational complexity and precision. In principle it should be possible to build an analysis of MS-DRT task systems on such an approach as well, but the complexity could become prohibitive.

Baruah (2012) has also proposed a variation of the standard mixed-criticality sporadic task model, in which the periods of sporadic tasks rather than their execution-time estimates are subject to uncertainties. A generalization by Ekberg and Yi (2014) covers the case where all parameters of the sporadic tasks may change, and the potential mode switches can be expressed as a directed acyclic graph instead of being linearly ordered. This generalized model is still much less expressive than the MS-DRT model.

Some limitations of sporadic mixed-criticality systems have also been addressed in other works recently. Santy et al. (2013) considered the transitioning back to lower criticality modes under both fixed-priority and EDF scheduling. Huang et al. (2014) additionally considered increasing the periods of low-criticality tasks rather than dropping them at a switch to a higher criticality mode when using EDF. Burns and Baruah (2013) instead looked at the analysis of fixed-priority scheduling when low-criticality tasks are allowed to decrease their execution-time budgets after a mode switch. Several authors (e.g., Buttazzo et al., 1998; Jan et al., 2013) have used elastic task models to let low-criticality tasks adapt their periods depending on the current load on the system. These solutions tend to be more specialized than the MS-DRT model presented in this paper, with which any number of complicated behaviors can be modeled on a per-task basis.

For a comprehensive review of the literature on mixed-criticality scheduling, we refer the reader to Burns and Davis (2015).
2 Model

In this section we describe the syntax and semantics of the MS-DRT task model. Some example tasks, focusing on a mixed-criticality interpretation of the semantics, are presented in Section 2.3. The task in Figure 1 helps to illustrate the syntax.

2.1 Syntax

An MS-DRT task system is formally defined by a finite set of tasks \( T = \{\tau_1, \tau_2, \ldots\} \) with an associated finite set of modes \( M(T) = \{\mu_1, \mu_2, \ldots\} \). An MS-DRT task \( \tau \in T \) is given by a triple \((V(\tau), E_{\text{cf}}(\tau), E_{\text{ms}}(\tau))\), defined as follows.

- \( V(\tau) \) is a set of vertices, representing job types.
- Each \( v \in V(\tau) \) is labeled by a triple \((e(v), d(v), \mu(v)) \in \mathbb{N}_0^2 \times M(T)\), representing worst-case execution time, relative deadline and mode of the corresponding job type, respectively.
- \( E_{\text{cf}}(\tau) \) is a set of directed edges representing possible task control flow, such that \( \mu(u) = \mu(v) \) for each \((u, v) \in E_{\text{cf}}(\tau)\). In the figures these edges are drawn as straight arrows.
- Each edge \((u, v) \in E_{\text{cf}}(\tau)\) is labeled with a minimum inter-release separation delay parameter \( p(u, v) \in \mathbb{N}_0 \).
- \( E_{\text{ms}}(\tau) \) is a set of directed edges representing possible mode switches, such that \( \mu(u) \neq \mu(v) \) for each \((u, v) \in E_{\text{ms}}(\tau)\). These edges are drawn as wiggly arrows.

We assume that each task \( \tau \in T \) satisfies the frame separation property, a generalization of the constrained deadlines concept for sporadic tasks. In other words, for each vertex \( u \in V(\tau) \) and \((u, v) \in E_{\text{cf}}(\tau)\) we have \( d(u) \leq p(u, v) \).

Note that, by the above definition, \( E_{\text{cf}}(\tau) \) and \( E_{\text{ms}}(\tau) \) are disjoint sets. Also, \((V(\tau), E_{\text{cf}}(\tau))\) is a directed graph with disjoint subgraphs for each mode of the task, and \((V(\tau), E_{\text{ms}}(\tau))\) is a directed multipartite graph (colorable with one color per mode). For convenience, let the subgraph in \((V(\tau), E_{\text{cf}}(\tau))\) corresponding to mode \( \mu_i \) be denoted

\[
\text{DRT}_{\mu_i}(\tau) \overset{\text{def}}{=} \left\{ v \in V(\tau) \mid \mu(v) = \mu_i \right\}, \left\{ (u, v) \in E_{\text{cf}}(\tau) \mid \mu(u) = \mu_i \right\}.
\]

2.2 Semantics

All tasks in an MS-DRT system run in the same mode at any particular time point, i.e., the modes are system wide. While running inside some mode \( \mu_i \), an MS-DRT task \( \tau \) behaves as an ordinary DRT task with graph \( \text{DRT}_{\mu_i}(\tau) \). That is, it releases a sequence of jobs that corresponds to some path (represented
Figure 1. An example task. The colors help reading, but carry no semantic information. In mode $\mu_1$, this task behaves as a simple two-vertex DRT task, releasing jobs at vertices $u$ and $v$ in some pattern. In $\mu_2$, the behavior is mostly sporadic with repeated job releases at $x$. The behavior at a mode switch from $\mu_1$ to $\mu_2$ depends on the state. If the latest job released in $\mu_1$ was at $v$, that job is dropped if it is still active upon the switch to $y$ (by setting its execution time budget to 0). Immediately after, $x$ can be visited and a new job be released. If the latest job in $\mu_1$ was instead at $u$, it is allowed to finish after the mode switch (its parameters are preserved in $w$) before the sporadic behavior at $x$ starts.

as a sequence of vertices) in $\text{DRT}_{\mu_i}(\tau)$, such that every vertex on the path matches one released job. More formally, a job of $\tau$ is defined by a pair $(r, v) \in \mathbb{R} \times V(\tau)$, representing a release time and a job type, respectively. A job sequence $[(r_1, v_1), (r_2, v_2), \ldots]$ is said to be generated by $\tau$ in $\mu_i$ if there is a path $(\pi_1, \pi_2, \ldots)$ through $\text{DRT}_{\mu_i}(\tau)$ such that for all $n$

1. $v_n = \pi_n$,
2. $r_{n+1} \geq r_n + p(\pi_n, \pi_{n+1})$.

A job $(r, u)$ has an execution-time budget equal to $e(u)$ and an absolute deadline equal to $r + d(u)$. The valid runtime behaviors of task $\tau$ in $\mu_i$ is to release any sequence of jobs that it can generate. Jobs may require execution time up to their budgets before they must finish, but may also finish earlier.

It is sometimes possible for the tasks in the task set $T$ to synchronously switch from their current mode $\mu_j$ to a new mode $\mu_i$. A mode switch from $\mu_j$ to $\mu_i$ is allowed if there is an outgoing mode-switch edge to the new mode from the latest job type of each task. More formally, it is allowed if and only if the latest released job of each task $\tau \in T$ is some $(r, u)$ and there exists an edge $(u, v) \in E_{\text{ms}}(\tau)$ such that $\mu(v) = \mu_i$.

When that mode switch occurs, each task synchronously switches to the new mode through one of its valid mode-switch edges $(u, v)$ and immediately updates its last released job $(r, u)$ correspondingly. In particular, the job $(r, u)$ is changed to become job $(r, v)$ as follows.
1. Its total execution-time budget is changed from \( e(u) \) to \( e(v) \), but is not replenished. If its remaining execution time is now less than or equal to 0, it is considered to have finished.

2. Then, its absolute deadline is changed to be \( r + d(v) \).

If a job has already finished before a mode switch it is never reactivated, even if its execution-time budget is increased. Jobs that are active during a mode switch are called *carry-over jobs*. A job is still eligible to become a carry-over job at the time point where its remaining execution-time budget reaches zero; this allows modeling of mode switches due to execution-time overruns.

After the mode switch, each task \( \tau \) can go on to release a new sequence of jobs \( [(r'_1, v'_1), (r'_2, v'_2), \ldots] \) in the new mode \( \mu_i \), as long as that sequence prepended by the updated job \( (r, v) \) can be generated by \( \tau \) in \( \mu_i \). By these semantics, minimum inter-release separation delays hold across mode switches. In other words, if the latest released job of \( \tau \) (active or not) was released at time \( r \) in a previous mode, then the first control-flow edge \( (v, w) \in E_{cf}(\tau) \) to be followed in the new mode can not be taken earlier than time \( r + p(v, w) \).

The model does not specify the origin of the events triggering mode switches, but rather just says that such events can arrive at any time. Any event-triggering scheme chosen by the system designer is then valid for the model. For example, mode-switch events can be emitted due to the run-time behavior of the tasks themselves, or due to execution-time overruns of jobs. They could also be the result of errors or faults, or come from external sources. A system may start with any mode as the initial one, and with any vertices with job types of that mode as the initial vertices of the tasks.\(^2\)

We define schedulability with some algorithm per mode of the system.

**Definition 2.1** (Schedulability). A mode \( \mu_i \in M(T) \) is \( A \)-schedulable if all jobs have finished latest at their deadlines while the system is in \( \mu_i \) and the jobs are executed in \( \mu_i \) by scheduling algorithm \( A \).

Note that the syntax and semantics of this model have intentionally been designed to be low-level, flat and suitable for timing analysis. Large and complex tasks could quickly become unwieldy for humans; we fully expect such tasks to be synthesized by tools rather than be manually crafted.

2.3 Examples

Here we present some simple example tasks, showing a few of the properties that can be modeled with the MS-DRT task model. The examples focus on mixed-criticality systems, but recall that the MS-DRT task model is not restricted to be interpreted as a model of such systems. An additional larger example is given in Appendix B.

\(^2\)In practice, systems will often have just a few initial states, but allowing it to start in any reachable state typically has no effect on schedulability.
**Example 2.1 (Dual-criticality tasks).** Figure 2 shows four tasks that are similar to ordinary mixed-criticality tasks, but some with additional semantics that can not be expressed in the original model. The intended interpretation is that the system would switch from a low-criticality mode (named LO) to a high-criticality mode (named HI) upon an execution-time overrun.

\( \tau_1 \) is equivalent to a high-criticality sporadic task, with period 28 and relative deadline 15, that gets its execution-time budget increased from 2 to 4 at a switch to the high-criticality mode (HI).

\( \tau_2 \) will instead drop any active job at a mode switch, and after a delay start a less intensive sporadic workload. It is like a low-criticality sporadic task that must provide a minimum quality of service also in the high-criticality mode, but holds back this service a short while to ease the transition be-
tween modes for the rest of the system. Recall that the inter-release separation constraints hold transparently across mode switches, so the extra dummy job at $v_3$ is introduced to ensure that $v_4$ is visited no earlier than 100 time units after the mode switch as opposed to 100 time units after the last job release at $v_1$. If $v_2$ instead was connected directly to $v_4$, a job could be released at $v_4$ immediately after the mode switch if enough time had passed since the last job release at $v_1$ in LO. Informally, the release of a dummy job at $v_3$ serves to reset the timer for the inter-release separation constraints.

$\tau_3$ will stop releasing new jobs after a mode switch, but must finish any active job that it has at that time; the time given to finish the last job is increased to 70 time units instead of the 30 time units that are normally given.

$\tau_4$ is a direct extension of a simple two-vertex DRT task to a high-criticality task with different execution-time estimates for the criticality levels.

$\tau'_1$

$\tau'_2$

Figure 3. Example tasks that can switch back to previous modes.
Example 2.2 (Cyclic criticality modes). Figure 3 shows two tasks that could be in a dual-criticality system where it is possible for the system to switch back to mode LO.

\( \tau'_1 \) exemplifies one possible way to model a high-criticality task. It releases jobs at most every 30 time units, and the execution-time estimate is 3 time units for the low-criticality mode (optimistic) and 6 time units for the high-criticality mode (pessimistic). The deadlines for the jobs are 30 time units after their release times, but for some vertices we have artificially decreased the relative deadlines to simulate the result of a tuning procedure. The intended interpretation is that \( \tau'_1 \) performs its normal mode of operation in \( u_1 \), moving to \( u_2 \) (and mode HI) upon an execution-time overrun. The carry-over job and the next job in HI will have the larger (original) deadline to provide extra slack during the mode transition, and the task eventually settles down in \( u_4 \), which has a smaller deadline parameter. The smaller deadline again provides some slack for the carry-over job should the system switch back to mode LO. The model allows the switch to LO to happen at any time, but the intended interpretation is that it should only happen if either a) the last visited vertex is \( u_2 \) or \( u_3 \) and the corresponding job is finished, or b) the last visited vertex is \( u_4 \) and the last job is not currently executing beyond the low-criticality budget of 3 time units. If there are several high-criticality tasks in the system, the intention is that switching back to LO should happen only when it is acceptable for all of them.

Essentially, the high-criticality behavior of \( \tau'_1 \) has been unrolled twice, creating vertices \( u_2 \) and \( u_3 \). The purpose is to allow the first two jobs in HI to have a different deadline and different semantics for switching back to LO. The number of times to unroll is a design-time choice for this type of task.

\( \tau'_2 \) is instead an example of how a type of low-criticality task can be modeled. Its normal mode of operation is in \( v_1 \). Upon a mode switch to HI (due to an execution time overrun of some high-criticality task), it drops any active job and becomes inactive. If the system switches back to LO, it additionally waits at least 20 time units before it begins to release new jobs at \( v_1 \) in order to ease the transition.

3 Analysis

In this section we introduce a structured methodology for analyzing the schedulability of MS-DRT task systems on preemptive uniprocessors. EDF analysis is presented in detail in this paper. The analysis is designed to consider each

---

3Some process of deadline tuning is essential for improving EDF-schedulability of mixed-criticality systems, and has previously been used for sporadic tasks (e.g., Baruah et al., 2011b, 2012; Ekberg and Yi, 2012, 2014; Easwaran, 2013; Zhang et al., 2014). Automatic deadline tuning is discussed further in Section 4.
mode of the system as independently as possible, abstracting the possible influences from preceding modes. For easy reference, a table of the notation used throughout this section is available in Appendix D.

**Definition 3.1** (Mode structure). The *mode structure* \( G(T) \) of an MS-DRT task system \( T \) is the directed graph \((V, E)\) where \( V = M(T) \) is the set of modes and \( E \) contains edges for the possible mode switches. That is, \((\mu_j, \mu_i) \in E \) if and only if each task \( \tau \in T \) has vertices \( u, v \) such that \((u, v) \in E_{ms}(\tau) \) and \( \mu(u) = \mu_j \) and \( \mu(v) = \mu_i \). Also, let the set of immediate predecessor modes to any mode \( \mu_i \) in \( G(T) \) be denoted \( \text{pred}_{G(T)}(\mu_i) \overset{\text{def}}{=} \{ \mu_j \mid (\mu_j, \mu_i) \in E \} \).

Note that \( G(T) \) contains no self-loops, but can otherwise be an arbitrary directed graph. Figure 4 shows the mode structures for the example task sets from the previous section.

![Figure 4. Mode structures of the tasks in Example 2.1 (left) and Example 2.2 (right).](image)

3.1 Overview of the EDF Analysis

The EDF analysis is based on computing demand bound functions for the task set. We define two different types of demand bound functions, covering different cases.

**Definition 3.2** (Internal demand bound functions). An *internal demand bound function* \( \text{idbf}_{\mu_i}(T, \ell) \) gives the maximum cumulative execution requirement of jobs from tasks in \( T \) that can be both released and have deadline in any time interval of length \( \ell \), during which the system is continuously in mode \( \mu_i \). The top of Figure 5 illustrates this type of demand bound function.

**Definition 3.3** (Transitional demand bound functions). A *transitional demand bound function* \( \text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell) \) gives the maximum cumulative execution requirement of jobs from tasks in \( T \) in any time interval of length \( \ell \), such that the interval starts at a mode switch from \( \mu_j \) to \( \mu_i \) and during which the system is continuously in mode \( \mu_i \). To be counted towards the cumulative execution requirement, a job must satisfy one of the following conditions.

1. Be released and have deadline inside the interval.
2. Be active at the time point of the mode switch and have deadline (the updated deadline, as seen in mode \( \mu_i \)) before the end of the interval.
In the latter case, only the workload that remains after the mode switch is counted, i.e., discounting any execution time that was done before the mode switch. The bottom of Figure 5 serves as an illustration.

\[ \mu_j \rightarrow \mu_i \]

**Figure 5.** Illustration of internal and transitional demand bound functions.

Internal demand bound functions can be computed directly using techniques from Stigge et al. (2011). Transitional demand bound functions offer a greater challenge. In order to determine the exact demand of carry-over jobs and actually compute a \( \text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell) \) we would have to consider, in great detail, the behavior of the system in \( \mu_j \), which in turn can depend on the mode preceding \( \mu_j \) and so on. A safe approximation of transitional demand bound functions is described in Section 3.3. For now, we define two predicates using the demand bound functions defined above, and show that they can be used to guarantee EDF-schedulability of a mode \( \mu_i \), considering a mode switch from another mode \( \mu_j \).

\[
\begin{align*}
S_{\text{EDF}}(T, \mu_i) & \overset{\text{def}}{=} \forall \ell \geq 0, \quad \text{idbf}_{\mu_i}(T, \ell) \leq \ell \\
S_{\text{EDF}}(T, \mu_j \rightarrow \mu_i) & \overset{\text{def}}{=} \forall \ell \geq 0, \quad \text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell) \leq \ell
\end{align*}
\]

**Lemma 3.1.** Mode \( \mu_i \) of MS-DRT task system \( T \) is EDF-schedulable when either of the following hold.

1. Mode \( \mu_i \) is the first mode the system is in and

\[ S_{\text{EDF}}(T, \mu_i) \]

2. Mode \( \mu_i \) is switched to from \( \mu_j \) and

\[ S_{\text{EDF}}(T, \mu_i) \land S_{\text{EDF}}(T, \mu_j \rightarrow \mu_i) \]
Proof. We prove the contrapositive. Assume that \( \mu_i \) was scheduled by EDF and that time point \( t_{\text{miss}} \) was the earliest time point in \( \mu_i \) at which some job \( J \) has missed its deadline. Let \([t_{\text{start}}, t_{\text{miss}}]\) be the busy period and \( t_{\text{switch}} \) the time point of the last switch from \( \mu_j \) to \( \mu_i \) if such a time point exists, or \( t_{\text{switch}} = \bot \) otherwise. The start of the busy period \( t_{\text{start}} \) is defined to be the earliest time point such that at all points in the interval \([t_{\text{start}}, t_{\text{miss}}]\) there was at least one active job with absolute deadline latest at \( t_{\text{miss}} \). Such a time point is guaranteed to exist. We now consider two cases.

First case: \( t_{\text{switch}} \notin [t_{\text{start}}, t_{\text{miss}}] \). In this case the busy period must be entirely in mode \( \mu_i \). By the definitions of the busy period and EDF, jobs with absolute deadline latest at \( t_{\text{miss}} \) were executed during the entire period. Again by definition, all jobs executed in the busy period were released inside it and so was the job \( J \) (regardless of whether it was executed or not). As not all jobs that were both released and had their deadline in the busy period finished inside it (in particular, job \( J \) did not), despite only those jobs being executed there, the total cumulative execution requirement of those jobs must exceed the length of the busy period. Because \( \text{idbf}_{\mu_i}(T, \ell) \) is the maximum cumulative execution requirement of jobs from tasks in \( T \) that are both released and have deadline in any interval of length \( \ell \) in \( \mu_i \), predicate \( \mathcal{S}_{\text{EDF}}(T, \mu_i) \) can not hold for \( \ell = t_{\text{miss}} - t_{\text{start}} \).

Second case: \( t_{\text{switch}} \in [t_{\text{start}}, t_{\text{miss}}] \). In this case the busy period can extend into previous modes. The part of the busy period that is inside \( \mu_i \) is \([t_{\text{switch}}, t_{\text{miss}}]\). During the whole of \([t_{\text{switch}}, t_{\text{miss}}]\), EDF scheduled only jobs with an absolute deadline latest at \( t_{\text{miss}} \), by the definition of the busy period. In addition, the jobs executed in \([t_{\text{switch}}, t_{\text{miss}}]\), as well as \( J \), must have been either released earliest at \( t_{\text{switch}} \) or been active at \( t_{\text{switch}} \) and carried over from \( \mu_j \). Because \( J \) did not finish inside the interval \([t_{\text{switch}}, t_{\text{miss}}]\), the cumulative execution requirement of those jobs (not counting workload finished before \( t_{\text{switch}} \)) must have exceeded the length of the interval. As \( \text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell) \) is the maximum cumulative execution requirement of exactly the above kind of jobs in an interval of length \( \ell \), starting at a mode switch from \( \mu_j \) to \( \mu_i \), \( \mathcal{S}_{\text{EDF}}(T, \mu_j \rightarrow \mu_i) \) can not hold for \( \ell = t_{\text{miss}} - t_{\text{switch}} \).

The schedulability guarantee provided by the above lemma is easily extended to cover all possible preceding modes.

**Corollary 3.2.** Mode \( \mu_i \) of MS-DRT task system \( T \) is EDF-schedulable if

\[
\mathcal{S}_{\text{EDF}}(T, \mu_i) \land \forall \mu_j \in \text{pred}_{G(T)}(\mu_i), \mathcal{S}_{\text{EDF}}(T, \mu_j \rightarrow \mu_i).
\]

3.2 Exact Formulation of Internal Demand Bound Functions

First we look at the internal demand bound functions. Because they only consider time intervals contained in a single mode and only jobs with both release time and deadline inside those intervals, they are equivalent to demand bound
The (beginning of) function $\text{idbf}_{\mu_i}^*(\tau_4, \ell)$ for $\tau_4$ from Example 2.1. The demand pairs for the paths through $\text{DRT}_{\mu_i}(\tau_4)$ are drawn as diamond-shaped points.

functions for ordinary DRT tasks, i.e., without mode switches. Such demand bound functions can be captured exactly by considering the paths in the single-mode subgraphs of each task. Every job sequence generated by a task $\tau$ while in any single mode $\mu_i$ corresponds to a path in the graph $\text{DRT}_{\mu_i}(\tau)$. Let $\Pi_{\mu_i}(\tau)$ denote the set of finite paths in $\text{DRT}_{\mu_i}(\tau)$. If $\pi$ is a path, let $\pi_n$ denote its $n$-th vertex, let $\pi_{n\cdots m}$ denote the (possibly empty) sub-path between and including the $n$-th and $m$-th vertices, and let $|\pi|$ denote its length in number of vertices.

For each path $\pi \in \Pi_{\mu_i}(\tau)$ we can calculate the maximum cumulative execution demand $\tilde{e}(\pi)$ of the job sequences corresponding to that path, as well as the minimum interval length $\tilde{d}(\pi)$ that can contain all the release times and absolute deadlines of such a job sequence.

$$
\tilde{e}(\pi) \overset{\text{def}}{=} \sum_{n=1}^{|\pi|} e(\pi_n) \quad (3)
$$

$$
\tilde{d}(\pi) \overset{\text{def}}{=} \sum_{n=1}^{|\pi|-1} p(\pi_n, \pi_{n+1}) + d(\pi_{|\pi|}) \quad (4)
$$

The pair $\langle \tilde{e}(\pi), \tilde{d}(\pi) \rangle$ is called a demand pair for path $\pi$. As is illustrated in Figure 6, the demand pairs for the paths in $\Pi_{\mu_i}(\tau)$ contain all the information needed to make a constructive formulation of the internal demand bound function for a single task $\tau$ (see Stigge et al., 2011):

$$
\text{idbf}_{\mu_i}^*(\tau, \ell) \overset{\text{def}}{=} \max\{\tilde{e}(\pi) \mid \pi \in \Pi_{\mu_i}(\tau) \land \tilde{d}(\pi) \leq \ell\} \quad (5)
$$

Because the tasks in a task set do not synchronize while the system remains in a single mode, any interleaving of job sequences from the tasks is possible. The sum of the internal demand bound functions for each task therefore ex-
actly matches the internal demand bound function for the task set, as defined in Definition 3.2.

\[ \text{idbf}_{\mu_i}(T, \ell) = \sum_{\tau \in T} \text{idbf}_{\mu_i}^*(\tau, \ell) \]  

(6)

It is shown by Stigge et al. (2011) how to efficiently compute all demand pairs that are relevant for establishing EDF-schedulability using a dynamic programming technique. This technique will also be used as the final step during the computation of approximated transitional demand bound functions. We now consider this approximation.

3.3 Approximation of Transitional Demand Bound Functions

Transitional demand bound functions can not, in general, be characterized exactly without a holistic analysis of the entire system. The complexity of such an analysis is likely prohibitive (recall that even for much simpler mixed-criticality sporadic tasks, an exact EDF analysis is yet to be found). Here we construct an approximation of transitional demand bound functions that is safe given the only assumption that the immediately preceding mode, i.e., the mode that is switched from, is schedulable with the scheduling algorithm used there. This may seem like a problematic assumption if there are cycles in the mode structure, but it does not actually cause any problems, as will be shown later in Theorem 3.7.

The approximation will be constructed in three steps that are outlined below.

1. We construct a function that is provably an upper bound on the true transitional demand bound function if the preceding mode is schedulable, but is impractical to compute.
2. From the first function we construct a second, simplified, function that is more practical to work with. It is a lower bound on the first function, but despite this it provably preserves safety in the schedulability analysis.
3. We construct a DRT task and show that its demand bound function is equal to the second function, and use the methods of Stigge et al. (2011) to compute it.

We begin by looking at individual carry-over jobs.

Approximating demand of carry-over jobs The carry-over jobs are the main issue to consider for transitional demand bound function \( t\text{dbf}_{\mu_j \rightarrow \mu_i}(T, \ell) \). Assume in the following that the preceding mode \( \mu_j \) is scheduled by algorithm \( A \) and is \( A \)-schedulable, meaning that no deadline can be missed in \( \mu_j \). It follows from the frame separation property (see Section 2) that there is at most one active job for each task at any time point in mode \( \mu_j \), and this is the job that was most recently released by the task. Each task \( \tau \) can then have at most one
carry-over job, and if \( \tau \) switched mode via edge \((u, v) \in E_{ms}(\tau)\), then the job that is carried over is of the type labeled on vertex \( u \). \(^4\) The job sequences we have to consider for each task therefore start with at most one carry-over job that is followed by a number of jobs that are both released and have deadline inside mode \( \mu_i \).

For each task \( \tau \), we must characterize the execution requirements of the (at most) single carry-over job. Let
\[
E_{\mu_j \rightarrow \mu_i}(\tau) \overset{\text{def}}{=} \{(u, v) \in E_{ms}(\tau) \mid \mu(u) = \mu_j \land \mu(v) = \mu_i\}
\]
denote the set of mode-switch edges that can take \( \tau \) from \( \mu_j \) to \( \mu_i \), and let
\[
\text{first}_{\mu_j \rightarrow \mu_i}(\tau) \overset{\text{def}}{=} \{w \mid (u, v) \in E_{\mu_j \rightarrow \mu_i}(\tau) \land (v, w) \in E_{cf}(\tau)\}
\]
denote the first vertices that can be visited in \( \mu_i \) via a control-flow edge after such a mode switch.

Clearly, each of the edges \((u, v) \in E_{\mu_j \rightarrow \mu_i}(\tau)\) that \( \tau \) may take on a mode switch could result in carry-over jobs of different parameters. However, also at any single edge \((u, v)\) the resulting carry-over job can be in any of a large number of different states. There are two important properties of carry-over jobs that are generally unknown:

1. At which time point, relative to the carry-over job’s scheduling window, does the mode switch occur?
2. How much of its execution time-requirement remains at that point?

Precise answers to those questions would likely require, as previously mentioned, a very detailed analysis of all possible behaviors of the system prior to a mode switch.

We assumed that the previous mode \( \mu_j \) is schedulable. It follows that for a job to be active at the mode switch and become a carry-over job, the switch must occur between its release time and absolute deadline in \( \mu_j \). For a carry-over job at edge \((u, v) \in E_{\mu_j \rightarrow \mu_i}(\tau)\), let \( x \) be the length of the time interval between the mode switch and the job’s absolute deadline in \( \mu_j \), as in Figure 7. We know that \( x \in [0, d(u)] \). Further, because \( \mu_j \) is schedulable, we know that if there had not been a mode switch (which is also a valid behavior according to the system model) the would-be carry-over job would have met its deadline in \( \mu_j \). The job’s remaining execution time budget in \( \mu_j \) at the time of the mode switch can therefore not exceed the length of the time interval until its deadline in \( \mu_j \), and is then at most \( \min(e(u), x) \). In the new mode \( \mu_i \), the total budget is changed to \( e(v) \) and the most that can remain of it immediately after the mode switch is
\[
e_{co}(u, v, x) \overset{\text{def}}{=} \max(0, e(v) - e(u) + \min(e(u), x)) .
\]

\(^4\)Even if the job was not released at \( u \), but in an even earlier mode, its job type must have been changed to the type of \( u \) prior to switching to mode \( \mu_i \).
We can also calculate the minimum length of a time interval starting at the mode switch, such that the carry-over job’s new deadline in $\mu_i$ is latest at the end of the interval, as

$$d_{co}(u, v, x) \overset{\text{def}}{=} \max(0, d(v) - d(u) + x).$$  \hfill (8)$$

Similarly, the minimum delay before a new control-flow edge $(v, w) \in E_{\text{cf}}(\tau)$ can be followed after the mode switch, resulting in the first job released in $\mu_i$, is

$$p_{co}(u, v, w, x) \overset{\text{def}}{=} \max(0, p(v, w) - d(u) + x).$$  \hfill (9)$$

A carry-over job at a mode-switch via $(u, v) \in E_{\mu_j \rightarrow \mu_i}(\tau)$ can therefore only add to the value of the transitional demand bound function $\text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell)$ if $d_{co}(u, v, x) \leq \ell$ for some $x \in [0, d(u)]$, and then with at most $e_{co}(u, v, x)$.

**Considering entire job sequences** Also jobs following the carry-over job (if there even is one) have to be considered for the transitional demand bound function. The execution requirements of the jobs following the carry-over job can be captured considering paths in $\text{DRT}_{\mu_i}(\tau)$, much in the same way as for the internal demand bound functions in Eq. (3) and (4).

We look first at the case where there is no carry-over job from task $\tau$. In this case, all the jobs from $\tau$ that can add to the cumulative execution requirement of $\text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell)$ are both released and have deadline in an interval of length
\( \ell \) that starts at the mode switch. Those jobs form a job sequence that corresponds to some path \( \pi \in \Pi_{\mu_i}(\tau) \), and a demand pair for that job sequence is simply

\[
\text{pair}_{\text{nco}}(\pi) \overset{\text{def}}{=} \langle \hat{e}(\pi), \hat{d}(\pi) \rangle.
\]

The first job to be released in such an interval must be of a type represented by one of the vertices in \( \text{first}_{\mu_j \rightarrow \mu_i}(\tau) \). The set of demand pairs for job sequences without carry-over jobs is therefore

\[
\text{nco}_{\mu_j \rightarrow \mu_i}(\tau) \overset{\text{def}}{=} \{ \text{pair}_{\text{nco}}(\pi) \mid \pi \in \text{Paths} \},
\]

where

\[
\text{Paths} = \{ \pi \in \Pi_{\mu_i}(\tau) \mid \pi_1 \in \text{first}_{\mu_j \rightarrow \mu_i}(\tau) \lor |\pi| = 0 \}.
\]

We now look at the case where there is a carry-over job from task \( \tau \). Each job sequence to consider from \( \tau \) then corresponds to some path \( \pi \in \Pi_{\mu_i}(\tau) \) where the first vertex is the carry-over job, i.e., where \( (u, \pi_1) \in E_{\mu_j \rightarrow \mu_i}(\tau) \) for some vertex \( u \). Again, the exact parameters of the carry-over job are unknown, but we know from Eq. (7) how to bound its remaining execution requirement when there are \( x \) time units left of its scheduling window in mode \( \mu_j \). Given such an \( x \in [0, d(u)] \), the cumulative execution requirement to consider for the entire job sequence is therefore at most

\[
\hat{e}_{\text{co}}(u, \pi, x) \overset{\text{def}}{=} e_{\text{co}}(u, \pi_1, x) + \hat{e}(\pi_2 \ldots |\pi|).
\]

Similarly, the minimal length of any time interval that starts at the mode switch and can contain the entire job sequence can be derived using Eq. (8) and (9). If the carry-over job is the only job in the sequence, the interval needs only be long enough to contain the carry-over job’s new deadline in \( \mu_i \). If there are other jobs in the sequence, the interval must be long enough to contain all their releases and deadlines, in addition to the minimum offset until the first of those jobs can be released. The minimal interval length is therefore

\[
\tilde{d}_{\text{co}}(u, \pi, x) \overset{\text{def}}{=} \begin{cases} 
\tilde{d}_{\text{co}}(u, \pi_1, x) & \text{if } |\pi| = 1, \\
p_{\text{co}}(u, \pi_1, \pi_2, x) + \tilde{d}(\pi_2 \ldots |\pi|) & \text{otherwise.}
\end{cases}
\]

Putting these together we can construct demand pairs for job sequences starting with a carry-over job as

\[
\text{pair}_{\text{co}}(u, \pi, x) \overset{\text{def}}{=} \langle \hat{e}_{\text{co}}(u, \pi, x), \tilde{d}_{\text{co}}(u, \pi, x) \rangle.
\]

We then consider all possible job sequences starting with a carry-over job, and all values of \( x \) for the carry-over job in each sequence. This way we define a safe approximation on the set of demand pairs for all job sequences starting with a carry-over job as

\[
\text{co}_{\mu_j \rightarrow \mu_i}(\tau) \overset{\text{def}}{=} \{ \text{pair}_{\text{co}}(u, \pi, x) \mid (u, \pi, x) \in \text{Vals} \},
\]

where

\[
\text{Vals} = \{ (u, \pi, x) \mid (u, \pi_1) \in E_{\mu_j \rightarrow \mu_i}(\tau) \land x \in [0, d(u)] \}.
\]
where \( \text{Vals} = \{(u, \pi, x) \mid \pi \in \Pi_{\mu_i}(\tau) \land (u, \pi_1) \in E_{\mu_j \rightarrow \mu_i}(\tau) \land x \in [0, d(u)]\} \).

Having safely approximated the demand pairs of all relevant job sequences, we can define an upper bound on the transitional demand bound function for a single task \( \tau \) in the same manner as in Eq. (5).\(^5\)

\[
tdbf_{\mu_j \rightarrow \mu_i}(\tau, \ell) \overset{\text{def}}{=} \max \{ e \mid (e, d) \in \text{Pairs} \land d \leq \ell \}, \tag{12}
\]

where \( \text{Pairs} = \text{nco}\mu_j \rightarrow \mu_i(\tau) \cup \text{co}\mu_j \rightarrow \mu_i(\tau) \).

By summing the over-approximated transitional demand bound functions of each \( \tau \in T \), we get an upper bound on the true transitional demand bound function of \( T \).

**Lemma 3.3.** If mode \( \mu_j \) is scheduled by algorithm \( A \) and is \( A \)-schedulable, then

\[
\forall \ell \geq 0, \sum_{\tau \in T} tdbf_{\mu_j \rightarrow \mu_i}(\tau, \ell) \geq tdbf_{\mu_j \rightarrow \mu_i}(T, \ell).
\]

**Proof.** By the reasoning above. \( \square \)

Note that this formulation has two sources of pessimism. The first is the potential over-approximation of the remaining execution time of carry-over jobs in Eq. (7). The second is that the summation of the single-task transitional demand bound function may combine worst cases for carry-over jobs that can not actually happen at the same time. Contrary, the jobs following a carry-over job in a job sequence are precisely captured by the demand pair technique and do not incur any additional pessimism.

**Reducing the number of demand pairs** In Lemma 3.3 it was shown that \( tdbf_{\mu_j \rightarrow \mu_i}(\tau, \ell) \) can be used as an upper bound on the transitional demand bound function, given that the preceding mode is schedulable. It is, however, impractical to compute using a dynamic programming technique such as that of Stigge et al. (2011). This is because it considers individual demand pairs for all possible sizes of the carry-over jobs’ remaining scheduling windows (i.e., all the different values of \( x \) in Eq. (11)).

To mitigate this, we create a new function \( tdbf^*_\mu_j \rightarrow \mu_i(\tau, \ell) \) considering only a single size for each carry-over job’s (remaining) scheduling window (i.e., a single value for \( x \)). We then show that the new function is safe to use for establishing schedulability even though it may at some points under-approximate \( tdbf_{\mu_j \rightarrow \mu_i}(\tau, \ell) \). We want to consider only the demand pairs that are, in a certain sense, the most problematic. For this we pick out a subset from the set of

\(^5\)Here it can be noted that if we would model a sporadic mixed-criticality task with MS-DRT, such as task \( \tau_1 \) in Example 2.1, the function \( tdbf_{LO \rightarrow HI}(\tau_1, \ell) \) would be equal to function \( dbf_{LO,H1}(\tau_1, \ell) \) from Eq. (8) in Ekberg and Yi (2014), although the formulation is completely different.
demand pairs in Eq. (11):

\[ \text{co}^*_{\mu_j \rightarrow \mu_i}(\tau) \overset{\text{def}}{=} \{ \text{pair}_{\text{co}}(u, \pi, e(u)) \mid (u, \pi) \in \text{Vals}^* \}, \]

where \( \text{Vals}^* = \{(u, \pi) \mid \pi \in \Pi_{\mu_i}(\tau) \land (u, \pi_1) \in E_{\mu_j \rightarrow \mu_i}(\tau)\} \).

The smaller set of demand pairs is used to define the final function as

\[ \text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell) \overset{\text{def}}{=} \max \{ e \mid \langle e, d \rangle \in \text{Pairs}^* \land d \leq \ell \}, \]

where \( \text{Pairs}^* = \text{nc} \text{co}_{\mu_j \rightarrow \mu_i}(\tau) \cup \text{co}^*_{\mu_j \rightarrow \mu_i}(\tau) \).

The following lemma shows that \( \text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell) \) preserves safety in schedulability analysis.

**Lemma 3.4.**

\[ \exists \ell_1 \geq 0, \sum_{\tau \in T} \text{tdbf}^*_{{\mu_j \rightarrow \mu_i}}(\tau, \ell_1) > \ell_1 \iff \exists \ell_2 \geq 0, \sum_{\tau \in T} \text{tdbf}^*_{{\mu_j \rightarrow \mu_i}}(\tau, \ell_2) > \ell_2 \]

**Proof.** The proof is in Appendix C.

We can now define new versions of the two predicates in (1) and (2).

\[ S^*_\text{EDF}(T, \mu_i) \overset{\text{def}}{=} \forall \ell \geq 0, \sum_{\tau \in T} \text{idbf}^*_{{\mu_i}}(\tau, \ell) \leq \ell \]

\[ S^*_\text{EDF}(T, \mu_j \rightarrow \mu_i) \overset{\text{def}}{=} \forall \ell \geq 0, \sum_{\tau \in T} \text{tdbf}^*_{{\mu_j \rightarrow \mu_i}}(\tau, \ell) \leq \ell \]

These predicates are safe replacements for their original counterparts:

**Lemma 3.5.** First, \( S^*_\text{EDF}(T, \mu_i) \iff S^*_\text{EDF}(T, \mu_i) \). Second, if mode \( \mu_j \) is scheduled by algorithm \( A \) and is \( A \)-schedulable, then

\[ S^*_\text{EDF}(T, \mu_j \rightarrow \mu_i) \implies S^*_\text{EDF}(T, \mu_j \rightarrow \mu_i) \]

**Proof.** From Eq. (6) we directly get \( S^*_\text{EDF}(T, \mu_i) \iff S^*_\text{EDF}(T, \mu_i) \). Lemmas 3.3 and 3.4 give us \( S^*_\text{EDF}(T, \mu_j \rightarrow \mu_i) \implies S^*_\text{EDF}(T, \mu_j \rightarrow \mu_i) \).

Our two main theorems follow. The first is about the EDF-schedulability of one mode given the schedulability of the possible preceding modes.
Theorem 3.6. Mode $\mu_i$ of MS-DRT task system $T$ is EDF-schedulable if each mode $\mu_j \in \text{pred}_{G(T)}(\mu_i)$ is scheduled by algorithm $A_j$ and is $A_j$-schedulable and if

$$S_{E_D F}^*(T, \mu_i) \land \forall \mu_j \in \text{pred}_{G(T)}(\mu_i), S_{E_D F}^*(T, \mu_j \rightarrow \mu_i).$$

Proof. By Corollary 3.2 and Lemma 3.5.

The second theorem shows that the same condition applied to all modes is sufficient to show the schedulability of the entire system if all modes are scheduled by EDF.

Theorem 3.7. If all modes of MS-DRT task system $T$ are scheduled by EDF, they are all EDF-schedulable if for all $\mu_i \in M(T)$,

$$S_{E_D F}^*(T, \mu_i) \land \forall \mu_j \in \text{pred}_{G(T)}(\mu_i), S_{E_D F}^*(T, \mu_j \rightarrow \mu_i).$$

Proof. By Lemmas 3.1 and 3.5 and by induction on the sequence of modes the system transitions through. The base case is the first mode $\mu_i$ that the system is in, which is guaranteed schedulability by $S_{E_D F}^*(T, \mu_i)$.

3.4 Efficiently Computing the Demand Bound Functions

We now look at how to efficiently evaluate the predicates $S_{E_D F}^*(T, \mu_i)$ and $S_{E_D F}^*(T, \mu_j \rightarrow \mu_i)$. There are two major challenges:

1. We cannot compute the functions $\text{idbf}_{\mu_i}^*(\tau, \ell)$ and $\text{tdbf}_{\mu_j \rightarrow \mu_i}^*(\tau, \ell)$ at all values of $\ell \geq 0$ because of the infinite domain.

2. Even for a given value of $\ell$, the number of possible paths through the graphs is generally exponential in $\ell$.

Fortunately, the above problems have been solved for regular DRT task sets in Stigge et al. (2011). For such task sets it was shown that if the utilization of the task set is bounded by some constant $c < 1$, it is enough to consider integer values of $\ell$ up to a pseudo-polynomial upper bound, providing a solution to the first point. For the second point, a path abstraction was introduced that enabled traversals of the graphs with dynamic programming. The computation of all relevant demand pairs could then be done in pseudo-polynomial time.

Because $\text{idbf}_{\mu_i}^*(\tau, \ell)$ is equal to the (regular) demand bound function of $\text{DRT}_{\mu_i}(\tau)$, the existing methods are directly applicable to the DRT task set \{DRT$_{\mu_i}(\tau)$ | $\tau \in T$\}, and can thus be used to evaluate $S_{E_D F}^*(T, \mu_i)$.

Mapping of functions to DRT tasks For $\text{tdbf}_{\mu_j \rightarrow \mu_i}^*(\tau, \ell)$ we have to do some more work. It would be possible to extend the methods of Stigge et al. (2011) to compute this function instead, but for brevity we opt to provide a construction of a DRT task whose demand bound function is exactly equal to
tdbf_{μ_j→μ_i}(τ, ℓ). This allows us to reuse the existing methods without the need to reprove their correctness in a new setting.\footnote{There are actually two minor technical differences remaining. One is that the original DRT task model assumes non-zero parameters (i.e., the labels on vertices and edges) while the DRT tasks we construct here may have zero-valued parameters. The other is that we restrict the considered paths to those that start at a subset of the vertices. The methods in Stigge et al. (2011) are easily extended to handle these differences, and we omit doing so here.}

The key observation that makes the construction possible is that for each mode-switch edge \((u, v) \in E_{μ_j→μ_i}(τ)\), we have reduced all the possible carry-over jobs considered for tdbf_{μ_j→μ_i}(τ, ℓ) to the equivalent of a single concrete job. The idea is to construct a new DRT task \(DRT_{μ_j→μ_i}(τ)\) by taking the graph \(DRT_{μ_i}(τ)\) and adding new vertices and edges to it representing the possible carry-over jobs. The details of the construction are as follows. Note that we omit to label vertices with modes as the resulting DRT tasks are non-modal.

1. For every \((u, v) \in E_{μ_j→μ_i}(τ)\), we add a vertex named \([u, v]\) after the corresponding edge to \(DRT_{μ_j→μ_i}(τ)\), and label the new vertex with the pair \((e_{co}(u, v, e(u)), d_{co}(u, v, e(u)))\).

2. For every newly added vertex \([u, v]\) and edge \((v, w) \in E_{cf}(τ)\), we add an edge \([([u, v], w)\) and label it with \(p_{co}(u, v, e(u))\).

It is evident that for every \((u, π) \in Vals^*\) used in Eq. (13), there is now a path \(π'\) in \(DRT_{μ_j→μ_i}(τ)\) such that \(π'_1 = [u, π_1]\) and \(π'_{2..|π'|} = π_{2..|π'|}\). By construction, we have

\[
⟨\tilde{ε}(π'), \tilde{d}(π')⟩ = pair_{co}(u, π, e(u)).
\]

The set of demand pairs for those paths in \(DRT_{μ_j→μ_i}(τ)\) that start at one of the newly added vertices therefore equal the set \(co_{μ_j→μ_i}(τ)\). The demand pairs in \(nco_{μ_j→μ_i}(τ)\) still correspond to the paths starting at one of the vertices in \(first_{μ_j→μ_i}(τ)\) (because the added vertices have no incoming edges they can never be a part of these paths and therefore don’t affect their set of demand pairs). We can then apply the methods from Stigge et al. (2011) to the task set \(\{DRT_{μ_j→μ_i}(τ) | τ \in T\}\), but restrict the considered paths to those starting at either one of the new vertices or a vertex in \(first_{μ_j→μ_i}(τ)\). It follows that the demand bound function computed in this way for \(DRT_{μ_j→μ_i}(τ)\) equal the function tdbf_{μ_j→μ_i}(τ, ℓ) over ℓ.

As an illustration, Figure 8 shows the DRT tasks constructed from the tasks in Example 2.2. It is easy to see that some of the vertices in the constructed tasks are redundant for the purposes of computing the demand bound function (e.g., \(u_2, [v_2, v_3]\) and practically all of \(DRT_{L0→H1}(τ'_2)\)), but this is not a problem because the dynamic programming graph traversal algorithm will ignore such vertices almost immediately.
Figure 8. The DRT tasks that are generated from the two MS-DRT tasks in Example 2.2. Valid start vertices are those drawn with solid lines; no path starting at a dashed vertex is considered when computing their demand bound functions.

**Complexity** The sizes of the constructed DRT task graphs are polynomial in the representation of the original MS-DRT task set, and the construction itself is a polynomial time operation. The values used as the labels on the constructed graphs never exceed the values used for labels on the MS-DRT task set. Note that $\text{DRT}_{\mu_j \rightarrow \mu_i}(\tau)$ has the same utilization as $\text{DRT}_{\mu_i}(\tau)$ because the added vertices are never part of a cycle. The evaluation of $S_{\text{EDF}}^*(T, \mu_i)$ or $S_{\text{EDF}}^*(T, \mu_j \rightarrow \mu_i)$ is therefore of pseudo-polynomial time complexity, following the results of Stigge et al. (2011), as long as the asymptotic utilization in $\mu_i$ is bounded by some constant $c < 1$.

To determine the EDF-schedulability of all modes in an MS-DRT task system $T$, as in Theorem 3.7, we need to perform $|V| + |E|$ such pseudo-polynomial time procedures, where $(V, E) = G(T)$. Note that traversals in
DRT$\mu_j \rightarrow \mu_i(\tau)$ are equivalent to traversals in DRT$\mu_i(\tau)$ after the first few vertices because the added vertices in DRT$\mu_j \rightarrow \mu_i(\tau)$ have no incoming edges. Much of the computation needed for establishing schedulability of a given mode $\mu_i$ (i.e., evaluating $S^*_{EDF}(T, \mu_i)$ and each $S^*_{EDF}(T, \mu_j \rightarrow \mu_i)$) therefore consists of repeated graph traversals in DRT$\mu_i(\tau)$ and can be combined into a single more efficient procedure.

4 Tuning

Tuning of the relative deadline parameters of tasks is an essential aspect of EDF-based scheduling of mixed-criticality systems. Various forms of deadline tuning have been successfully applied to mixed-criticality sporadic task sets before (see, e.g., Baruah et al., 2011b, 2012; Ekberg and Yi, 2012, 2014; Easwaran, 2013; Zhang et al., 2014). The key idea is to artificially decrease the deadline of a job in one mode, and then to revert back to the larger deadline if it would become a carry-over job. In this way it is provided with extra slack time during the transitional period following a mode switch. Note that if we never assign a value to a deadline parameter that is larger than the original value, we are still keeping the timing constraints given by the system designer.

A similar tuning approach can be used to improve the EDF-schedulability of an MS-DRT task system (regardless of whether it is interpreted as a mixed-criticality system or not). Consider the case where task $\tau$ has a carry-over job at some mode-switch edge $(u, v) \in E_{\mu_j \rightarrow \mu_i}(\tau)$. The carry-over job’s timing properties in mode $\mu_i$ are characterized for the EDF-analysis by Eq. (7)–(9). If we were to decrease the relative deadline parameter $d(u)$ by some value $\delta$, then $e_{co}(u, v, x)$ would remain the same (for a given $x$) while $d_{co}(u, v, x)$ and $p_{co}(u, v, w, x)$ would both increase by up to $\delta$. In other words, the carry-over job would be easier to schedule after a mode switch, at the expense of making jobs from vertex $u$ harder to schedule in mode $\mu_j$ due to the smaller deadline.

Finding valid values for all relative deadlines such that some condition is met (e.g., so that all predicates in Theorem 3.7 hold) is a form of constraint satisfaction problem. There are many ways of attempting to solve such a problem. For example, in Ekberg and Yi (2012, 2014) a heuristic search algorithm tunes each deadline parameter individually while a condition based on demand bound functions is checked. For MS-DRT task sets that are somewhat similar to sporadic task sets and have a mode structure that is a DAG (like the tasks in Example 2.1), we believe that a heuristic tuning approach inspired by that in Ekberg and Yi (2014) should work well. For more complicated tasks and cyclic mode structures, new tuning heuristics would be required. We consider the design of tuning heuristics to be out of scope of this paper. We instead outline a general method of implementing tuning procedures efficiently by eliminating redundant recomputations after a parameter change.
4.1 Avoiding Recomputations

If we change the relative deadline parameter of some vertex $u \in V(\tau)$, where $\mu(u) = \mu_i$, that change can affect all the demand bound functions on mode $\mu_i$, i.e., $\text{idbf}^*_{\mu_i}(\tau, \ell)$ and $\text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell)$ for each predecessor mode $\mu_j$. If jobs of type $u$ can be carried over to another mode $\mu_k$, then the parameter change can also affect $\text{tdbf}^*_{\mu_i \rightarrow \mu_k}(\tau, \ell)$. These are demand bound functions defined by the demand pairs of paths in one of the graphs $\text{DRT}_{\mu_i}(\tau)$, $\text{DRT}_{\mu_j \rightarrow \mu_i}(\tau)$ or $\text{DRT}_{\mu_i \rightarrow \mu_k}(\tau)$.

It is wasteful to recompute those demand bound functions from scratch every time a parameter value is changed. Clearly, the change to $d(u)$ only affects demand pairs corresponding to paths that contain $u$ in some form. That is, paths that contain at least one of the vertices $u$, $[v, u]$ or $[u, v]$ for some $v$. In addition, the demand pair $(\hat{e}(\pi), \hat{d}(\pi))$ of such a path $\pi$ is only affected by the change to $d(u)$ if either $\pi_1$ is $[u, v]$ or $\pi_{|\pi|}$ is $u$ or $[v, u]$. Changes to deadlines of intermediate vertices in $\pi$ do not affect its demand pair. In fact, given any path $\pi$ we only need to know the identities of vertices $\pi_1$, $\pi_2$ and $\pi_{|\pi|}$ to update its demand pair to be valid for a new value of parameter $d(u)$ for any arbitrary vertex $u \in V(\tau)$. (The identity of $\pi_2$ is needed to calculate the minimum separation between $\pi_1$ and $\pi_2$ with Eq. (9).) See Figure 9 for an illustration.

![Figure 9](image_url)  
**Figure 9.** At most three vertices of path $\pi$ must be considered when updating its demand pair upon a deadline change. For this path, a decrease of $d(z)$ results in a decrease of $\hat{d}(\pi)$, and a decrease of $d(u)$ may instead increase $\hat{d}(\pi)$. Changes to the deadline of other vertices, such as $d(y)$, do not affect the demand pair.

We propose to perform the tuning at the level of the path abstractions used in the graph traversals. The path abstraction used in Stigge et al. (2011) is a triple $\langle e, d, v \rangle$, consisting of a demand pair $\langle e, d \rangle$ extended with a single vertex $v$; it represents all paths that end in $v$ and have demand pair $\langle e, d \rangle$. It forms the basis for the dynamic programming based graph traversal because longer paths can be created from it by just replacing $v$ with some successor vertex $u$ and updating the values $e$ and $d$ accordingly. After generating all (relevant) path abstractions, the set of demand pairs is easily extracted.

We extend the path abstraction to contain also the first and second vertices of the paths it represent, so it becomes a triple $\langle e, d, (u, v, w) \rangle$. It then represents all paths with demand pair $\langle e, d \rangle$ starting with vertices $u$ and $v$ and ending in $w$. The graph traversal proceeds like before by replacing $w$ and updating $e$ and $d$. With the extension there are clearly more path abstractions that can be created in total, but from Stigge et al. (2011) it is clear that they are...
still at most pseudo-polynomially many. After performing the graph traversals once, we can perform deadline tuning directly on the sets of path abstractions generated for each induced DRT graph. There are many further optimizations that can be applied, but the above provides a proof of concept that efficient deadline tuning is possible.

5 Conclusions

We have presented the MS-DRT task model, which combines complex job arrival patterns with state-based mode changes. The mode-switching protocol is of a mixed-criticality style, meaning that parameters of active jobs may be immediately changed upon a mode switch. A consequence of this is that the task model generalizes both previous graph-based and mixed-criticality (sporadic) task models. The model can express some features of mixed-criticality systems that are lacking in the standard mixed-criticality task models. Being fairly general, it may also be useful as a timing model for other state-based systems with various types of modes, such as some used in model-based design tools.

There are several ways in which the task model can be extended. For example, one can add a language for describing conditions on how and when different mode switches can occur. Another extension is to allow arbitrary deadlines, with the consequence that each task may have several carry-over jobs at once. Also the latter would require some syntactic changes in order to specify how each of the possibly several carry-over jobs should be changed, for example by labeling job-type mappings on the mode-switching edges. A topic of further study is to figure out which extensions to the model that are both useful and analyzable with reasonable efficiency and precision.

We have described a structured EDF-schedulability analysis for the proposed task model. The analysis does not require that all modes are scheduled by EDF, but only assumes that other modes are schedulable with whatever scheduling algorithm is used there. As a future work, it might be possible to adapt the work of Stigge and Yi (2013) on fixed-priority schedulability analysis for DRT task systems to the MS-DRT setting.

Appendix A Some Preliminary Experiments

A.1 Motivation

It is possible to model many different types of systems using MS-DRT, but at this stage we find it generally difficult to quantitatively evaluate the effectiveness of both the modeling formalism itself and the proposed schedulability analysis, because there is little to directly compare with. Here we try to illustrate the effectiveness of our approach on a restricted set of systems that address a common concern voiced about mixed-criticality scheduling, namely
the usual assumption that low-criticality tasks are dropped upon a switch to a higher criticality mode. Often, it is instead desirable to guarantee a minimal quality of service (QoS) to the low-criticality tasks even after a mode switch.

There have been some attempts to solve this in the context of sporadic mixed-criticality tasks by allowing low-criticality tasks to continue executing in the higher criticality mode, but with new parameters (see, e.g., Burns and Baruah, 2013; Ekberg and Yi, 2014). In these works, low-criticality tasks are essentially treated the same as high-criticality tasks in the sense that they are immediately given new parameters upon a mode switch. Contrary to the high-criticality tasks that typically get worsened parameters (e.g., increased execution times) in the new mode, low-criticality tasks are changed to have a smaller impact on the system, for example by decreasing their execution times or increasing their periods. The motivation for doing so is that if such a system is schedulable, it will provide some QoS guarantees for low-criticality tasks even in the higher criticality mode.

However, we argue that using such an approach will unnecessarily limit schedulability. The reason is that it makes the transitional periods after mode switches harder to successfully schedule. The major challenge in guaranteeing the schedulability of a mixed-criticality system is to show that all jobs that are active shortly after a mode switch will meet their deadlines, in particular the carry-over jobs. If also the low-criticality tasks can have carry-over jobs, ensuring schedulability becomes significantly harder. At the same time, we argue that low-criticality tasks do not need to be treated the same as those of higher criticality simply to provide some QoS in the new mode. The mode switch protocol used for the high-criticality tasks—to immediately change the parameters, including those of active jobs—is, after all, quite extreme. It was designed to make sure that critical tasks will continue to function without any delay whatsoever even in the face of invalid parameter estimates. Instead, by simply pausing low-criticality tasks for a short period of time after a mode switch, before restarting them with less intensive parameters, we can achieve almost the same QoS guarantees without sacrificing schedulability.

Tasks that pause activity for a while after a mode switch are easily modeled with MS-DRT, for example as task $\tau_2$ of Example 2.1. In this evaluation we will compare three different types of low-criticality tasks: those without QoS guarantees in the high-criticality mode; those with basic guarantees as proposed by Burns and Baruah (2013) and Ekberg and Yi (2014); and those that introduce a small delay after a mode switch before restarting. These different types of tasks are shown in Figure 10, together with the standard type of high-criticality task that they will be mixed with.

A.2 Task Set Generation

To evaluate the differences between these approaches, we generate random task sets where the low-criticality tasks are of one of the three different types,
Type HI
Ordinary high-criticality task 
\((e' \geq e)\).

Type LO-no-QoS
Ordinary low-criticality task without QoS guarantee.

Type LO-basic-QoS
Low-criticality task with basic QoS guarantee 
\((e' \leq e, p' \geq p, d' \geq d)\).

Type LO-delay-QoS
Low-criticality task with QoS guarantee after initial delay \(\delta\) 
\((e' \leq e, p' \geq p, d' \geq d, \delta \geq 0)\).

Figure 10. A standard high-criticality task and three different types of low-criticality tasks.

and compare their EDF-schedulability according to the analysis presented in this paper. First, we define a few constants used for the task set generation, namely

\[
Pr_{HI} = 0.5, \quad \text{the probability of each task to be of high criticality,}
\]

\[
e_{LO}^{max} = 10, \quad \text{the maximum execution time budget in mode LO,}
\]

\[
p_{LO}^{max} = 200, \quad \text{the maximum period in mode LO,}
\]

\[
e_{HI}^{fact} = 4, \quad \text{the maximum factor of execution time increase in HI (for HI-tasks),}
\]

\[
e_{LO}^{fact} = 0.5, \quad \text{the minimum factor of execution time decrease in HI (for LO-tasks),}
\]

\[
p_{LO}^{fact} = 2, \quad \text{the maximum factor of period increase in HI (for LO-tasks).}
\]

The first three constants were chosen somewhat arbitrarily. The value for \(e_{HI}^{fact}\) was set to 4 because a difference of up to four times between, say, measurement-based and static analysis-based WCET estimates seems fairly realistic for complex code and hardware platforms. To balance this, low-criticality tasks may be limited to as little as half their ordinary execution time \((e_{LO}^{fact})\) and double their periods \((p_{LO}^{fact})\) in the high-criticality mode.

Each task set is generated with a target utilization \(U^*\) in mind. A generated task set \(T\) is considered valid only if \(U_{avg}(T) \in [U^* - 0.005, U^* + 0.005]\).
where
\[
U_{\text{avg}}(T) \overset{\text{def}}{=} \sum_{\mu_i \in M(T)} \frac{U(T, \mu_i)}{|M(T)|}
\]
is the average utilization of \( T \) and \( U(T, \mu_i) \) is the asymptotic utilization of \( T \) in mode \( \mu_i \). In addition, a task set \( T \) is considered valid only if its utilization is at most 0.99 in each mode, and each mode is EDF-schedulable in its steady-state. The former is a practical restriction to limit analysis time, the latter a restriction to the interesting cases where task sets are not trivially unschedulable no matter how the mode changes are handled. Generated task sets that are deemed invalid are simply discarded and new ones are generated instead. The task sets are always generated three at a time, each with low-criticality tasks of a certain type, the parameters of the tasks in each of the sets are kept identical where applicable. The details of the task set generation are found in Algorithm 1. Note that the values of the \( \delta \)-labels for the tasks of type LO-delay-QoS are not determined by Algorithm 1, we will instead search for suitable values as part of the experiment. Also note that task sets with type LO-no-QoS tasks can have smaller average utilization than the other two task sets, \( U^\ast \) is only compared against the average utilization of the task sets with QoS tasks.

A.3 Evaluation

When determining the EDF-schedulability of the task sets, we use the deadline tuning algorithm \texttt{TuneSystem} from Ekberg and Yi (2014). It is directly applicable to the task sets with low-criticality tasks of type LO-no-QoS and LO-basic-QoS (indeed, even the schedulability analysis in Ekberg and Yi (2014) is equivalent to the one in this paper for such tasks). For task sets with type LO-delay-QoS tasks, we use the naive extension of \texttt{TuneSystem} that only attempts to tune deadlines of high-criticality tasks.

We still have to determine the values of the \( \delta \) parameters of the tasks of type LO-delay-QoS. For simplicity, we assume that all such tasks in a task set \( T_{\text{delay}} \) have the same value for their \( \delta \) parameter. For each such task set we do a binary search on the ordered set \( \{0, \ldots, p_{\text{LO}}^{\text{max}}\} \) to find the minimal value for \( \delta \) with which \( T_{\text{delay}} \) is deemed EDF-schedulable. Figure 11 shows the acceptance ratios of the various types of task sets, without QoS, with basic QoS and with delayed QoS. For the task sets with delayed QoS, acceptance ratios are plotted for when the \( \delta \) parameters are bounded by some different constants. Each data point is based on 10,000 randomly generated task sets.

From Figure 11 we can see that even for relatively small delays, less than half of the maximum period, acceptance ratios are practically the same for task sets with QoS as for those without. In contrast, task sets with basic QoS that use the same mode-switching logic for both high- and low-criticality tasks
Figure 11. Acceptance ratios for the three different types of task sets.

Figure 12. The minimum values for $\delta$ needed to make the task sets schedulable.

have significantly lower acceptance ratios. Even when $\delta = 0$ there was an increase in acceptance ratio compared to the basic QoS. The reason is that it is easier to schedule tasks that drop active jobs at a mode switch and immediately release new ones than it is to schedule tasks with carry-over jobs.

Figure 12 shows the average value for the $\delta$ parameter that was necessary to make the task sets $T_{\text{delay}}$ schedulable, when such a value could be found in $\{0, \ldots, p_{\text{max}}^{\text{LO}}\}$. The error bars indicate the standard deviation of the sample. It is clear from the figure that even for very large utilizations, small $\delta$ parameters tend to be sufficient.
Algorithm 1: Algorithm for generating random task sets.

1 Function generate-task-sets($U^*$, $Pr_{HI}$, $e_{max}^{LO}$, $p_{max}^{LO}$, $e_{fact}$, $p_{fact}$):
2 $T_{no}, T_{basic}, T_{delay} \leftarrow \emptyset, \emptyset, \emptyset$
3 repeat
4   with probability $Pr_{HI}$ do
5      $(e, d, p, e') \leftarrow$ get-hi-task-params($e_{max}^{LO}$, $p_{max}^{LO}$, $e_{fact}$)
6      $\tau_{HI} \leftarrow$ Type HI task with parameters $(e, d, p, e')$
7      $T_{no} \leftarrow T_{no} \cup \{\tau_{HI}\}$
8      $T_{basic} \leftarrow T_{basic} \cup \{T_{HI}\}$
9      $T_{delay} \leftarrow T_{delay} \cup \{\tau_{HI}\}$
10   otherwise do
11      $(e, d, p, e', d', p') \leftarrow$ get-lo-task-params($e_{max}^{LO}$, $p_{max}^{LO}$, $e_{fact}$, $p_{fact}$)
12      $\tau_{LO}^{no} \leftarrow$ Type LO-no-QoS task with parameters $(e, d, p)$
13      $T_{no} \leftarrow T_{no} \cup \{\tau_{LO}^{no}\}$
14      $\tau_{LO}^{basic} \leftarrow$ Type LO-basic-QoS task with parameters $(e, d, p, e', d', p')$
15      $T_{basic} \leftarrow T_{basic} \cup \{\tau_{LO}^{basic}\}$
16      $\tau_{LO}^{delay} \leftarrow$ Type LO-delay-QoS task with parameters $(e, d, p, e', d', p')$
17      $T_{delay} \leftarrow T_{delay} \cup \{\tau_{LO}^{delay}\}$
18      if is-valid-task-set($T_{basic}$) then
19         $T_{no}, T_{basic}, T_{delay} \leftarrow \emptyset, \emptyset, \emptyset$
20 until $U^* - 0.005 \leq U_{avg}(T_{basic}) \leq U^* + 0.005$
21 return ($T_{no}, T_{basic}, T_{delay}$)

22 Function get-hi-task-params($e_{max}^{LO}$, $p_{max}^{LO}$, $e_{fact}$):
23 $e \leftarrow$ random sample from $U\{1, \ldots, e_{max}\}$ // Discrete uniform distribution
24 $e_{fact} \leftarrow$ random sample from $U[1, e_{fact}]$ // Continuous uniform distribution
25 $e' \leftarrow e \cdot e_{fact}$
26 $p \leftarrow$ random sample from $U\{e', \ldots, p_{max}\}$
27 $d_{fact} \leftarrow$ random sample from $U[0, 1]$  
28 $d \leftarrow \max(e', [p \cdot d_{fact}])$
29 return $(e, d, p, e')$

30 Function get-lo-task-params($e_{max}^{LO}$, $p_{max}^{LO}$, $e_{fact}$, $p_{fact}$):
31 $e \leftarrow$ random sample from $U\{1, \ldots, e_{max}\}$
32 $e_{fact} \leftarrow$ random sample from $U[e_{fact}, 1]$
33 $e' \leftarrow e \cdot e_{fact}$
34 $p \leftarrow$ random sample from $U\{e, \ldots, p_{max}\}$
35 $p_{fact} \leftarrow$ random sample from $U[1, p_{fact}]$
36 $p' \leftarrow p \cdot p_{fact}$
37 $d_{fact} \leftarrow$ random sample from $U[0, 1]$  
38 $d \leftarrow \max(e, [p \cdot d_{fact}])$
39 $d' \leftarrow \max(e', [p' \cdot d_{fact}])$
40 return $(e, d, p, e', d', p')$

41 Function is-valid-task-set($T$):
42 if $U(T, LO) \leq 0.99 \land U(T, HI) \leq 0.99 \land S_{EDF}(T, LO) \land S_{EDF}(T, HI)$ then
43   return true
44 return false
Appendix B  A Larger System Example

Burns (2014) recently attempted to unify various notions of mode changes that has been used in the literature, in particular various general mode changes and criticality mode changes. He provides a high-level description of an example cruise-control system in a car that is complicated by having, at the same time, different types of modes and mode change protocols. As another motivation for MS-DRT, we outline in this section how it can be used to model that example system. First, we briefly summarize the terminology of Burns (2014), starting with the three main types of modes that he identifies.

Normal functional modes are modes that are switched between as part of the regular operation of the system.

Exceptional functional modes are modes that are entered as a response to some rare events.

Degraded functional modes are modes entered as a consequence of some error or fault in the system, where some normal functionalities may be shed in order to give priority to safety-critical functions.

Further, Burns characterizes three main types of mode changes:

Immediate mode changes cause old jobs to be suspended or aborted, and new jobs from the new mode to be started immediately.

Bounded mode changes wait until there are no active jobs from the old mode and then switch cleanly to the new mode.

Phased mode changes let old jobs finish, and new jobs may be released within some bounded time, even if all old jobs have not finished.

Transitions for the above three types of mode changes can be modeled with MS-DRT, for example as in Figure 13. Note that transitions for immediate and bounded mode changes are modeled in the same way, but with different interpretations of the semantics. For immediate transitions, we interpret the mode switch event as being propagated immediately, causing any active job to be dropped (by setting its execution time budget to 0 in vertex $v$) and a new job to be released immediately at $w$. On the other hand, for bounded
transitions we interpret the mode switch event to occur when all old jobs have finished, at which point no job from \( u \) is dropped at the transition to \( v \). For the schedulability analysis, these two scenarios look identical. In a phased transition, old active jobs are brought along to the new mode (though we allow changing their parameters in the process), and new jobs may be released before all of them are finished.

The cruise-control system described by Burns consists of two normal functional modes, standby (SB) and speed control (SC), and one exceptional mode, collision avoidance (CA). According to Burns, transitions between SB and SC should be either bounded or phased, and transitions from either of them to CA should be immediate. We pick phased transitions between SB and SC to make the example more interesting.

In addition, the system software is partitioned into two criticality levels, called SIL2 and SIL4. Code for SIL4 has two WCET estimates, one lower measurement-based estimate that is valid at SIL2 and one higher static-analysis based valid at SIL4. If at any time some WCET estimate at SIL2 turns out to be invalid, the system should enter some form of degraded mode where more time is given to the most critical tasks at the expense of the less critical. For the critical tasks, this would imply some kind of phased transition where execution-time budgets of active jobs get immediately inflated. In effect, we get six modes in total, the three modes SB, SC and CA using SIL2 WCET assumptions, and degraded versions of the same modes valid at SIL4. We call the modes SB\(_2\), SC\(_2\) and CA\(_2\) in SIL2, and SB\(_4\), SC\(_4\) and CA\(_4\) in SIL4. With these names we can form the mode structure of the system as in Figure 14.

![Figure 14. Mode structure of the cruise-control system.](image)

Recall that MS-DRT does not impose any minimum separation delays between mode switches, other than what is explicitly put into the tasks themselves. This means that the schedulability analysis described earlier is valid for all possible sequences of mode switches, including complex situations such...
as a transition to $CA_2$ happening in the middle of a phased transition between $SB_2$ and $SC_2$, closely followed by a transition to $CA_4$. This was identified by Burns as a difficult problem.

In the system description given for this example, one particular task was also outlined. This is a sporadic task responsible for proximity analysis. It is stated that it should run in all three modes, but have a smaller period in $CA$. We assume that it is meant to have the same parameters in both $SB$ and $SC$. Additionally, we assume that it belongs to the higher criticality level (SIL4), and therefore should run also in the degraded modes with a larger execution time budget. In Figure 15 we have modeled this task. As no parameter values were given by Burns, we have arbitrarily picked some. We picked a WCET of 4 time units at SIL2 and 7 time units at SIL4. For the period we chose 50 time

Figure 15. The proximity analysis task.
units in the various SB and SC modes, and 30 time units in the CA modes. The delay associated with the phased transition between SB and SC is set to 100. All deadlines are implicit.

This particular task was easy to model with only two vertices per mode. One work vertex per mode, with the name of the mode superscripted by “w”, captures the sporadic behavior of the task in that mode. Another gate vertex, superscripted instead by “g”, captures mode transition logic between modes at either SIL2 or SIL4 in the manner showed in Figure 13. When switching from some mode at SIL2 to the corresponding one at SIL4 (e.g., from SB₂ to SB₄) the mode switching logic is that control is just moved to a mirrored version of the same vertex in the higher criticality level. We have intentionally omitted a mode switching edge from CA₂⁵ to CA₄⁵ with the interpretation that no time ever passes before moving on from CA₂⁵ to CA₂ʷ.

In Burns’ description, there is no mention of mode changes being possible in order to go back from a CA mode to a SB or SC mode, but this seems like a desirable feature and may have been unintentionally omitted from the description. Adding this feature to a task such as the one in Figure 15 is not difficult. Additionally, it would be possible to model mode switches from a SIL4 mode back to the corresponding SIL2 mode, resulting in the strongly connected mode structure in Figure 16. The easiest way to model this would be with bounded transitions and the interpretation that such a mode switch can happen at any idle time, but it is also possible to model something more elaborate, e.g., as in Example 2.2.

![Figure 16. An extended mode structure that is strongly connected.](image)

We note that the task in Figure 15 is quite large. Manually crafting such tasks certainly puts a burden on the system designer and would likely be error-prone. We envision that large tasks in practice should be synthesized by some model-based design tool or, at least, be manually modeled using some higher-level representation with syntactic sugar for common constructs.
Appendix C  Proof of Lemma 3.4

To prove Lemma 3.4, we first define a relation on demand pairs and prove an auxiliary lemma.

Definition C.1 (Cover relation). A demand pair \( \langle e, d \rangle \) covers another demand pair \( \langle e', d' \rangle \), denoted \( \langle e, d \rangle \trianglerighteq \langle e', d' \rangle \), if and only if

\[
e \geq e' \quad \text{and} \quad e - e' \geq d - d'.
\]

Figure 17 illustrates the cover relation. The intuition behind the cover relation is that a demand pair should cover all other demand pairs that are no more problematic from a scheduling point of view.

![Figure 17](image)

**Figure 17.** The demand pair (3, 5) covers all pairs in the shaded area.

The subset of demand pairs used to define \( \text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell) \) covers the set of demand pairs used to define \( \text{tdbf}^\text{ub}_{\mu_j \rightarrow \mu_i}(\tau, \ell) \), as shown in the following lemma.

**Lemma C.1.** For each demand pair \( \langle e, d \rangle \in \nco_{\mu_j \rightarrow \mu_i}(\tau) \cup \co_{\mu_j \rightarrow \mu_i}(\tau) \) there exist an \( \langle e^*, d^* \rangle \in \nco_{\mu_j \rightarrow \mu_i}(\tau) \cup \co^*_{\mu_j \rightarrow \mu_i}(\tau) \) such that \( \langle e^*, d^* \rangle \trianglerighteq \langle e, d \rangle \).

Proof. The lemma trivially holds for each \( \langle e, d \rangle \in \nco_{\mu_j \rightarrow \mu_i}(\tau) \) because the cover relation is reflexive.

We consider instead a demand pair \( \langle e, d \rangle \in \co_{\mu_j \rightarrow \mu_i}(\tau) \). From Eq. (11) it is evident that \( \langle e, d \rangle = \text{pair}_{\co}(u, \pi, x) \) for some \( (u, \pi, x) \in \text{Vals} \). We split the proof into three cases.
Case 1 ($x \geq e(u)$):
Let $\langle e^*, d^* \rangle = \text{pair}_{\text{co}}(u, \pi, e(u))$. Clearly, $\langle e^*, d^* \rangle \in \text{co}_{\mu_j \rightarrow \mu_i}(\tau)$. We calculate

\[
e = \tilde{e}_{\text{co}}(u, \pi, x) \\
= e_{\text{co}}(u, \pi_1, x) + \tilde{e}(\pi_2 \ldots |\pi|) \\
= e_{\text{co}}(u, \pi_1, e(u)) + \tilde{e}(\pi_2 \ldots |\pi|) \\
= \tilde{e}_{\text{co}}(u, \pi, e(u)) \\
= e^*
\]

and $d = \tilde{d}_{\text{co}}(u, \pi, x) \geq \tilde{d}_{\text{co}}(u, \pi, e(u)) = d^*$. It follows that $\langle e^*, d^* \rangle \trianglerighteq \langle e, d \rangle$.

Case 2 ($x \leq e(u) - e(\pi_1)$):
Let $\langle e^*, d^* \rangle = \text{pair}_{\text{nco}}(\pi_2 \ldots |\pi|)$. Because either $|\pi_2 \ldots |\pi| = 0$ or $\pi_2 \in \text{first}_{\mu_j \rightarrow \mu_i}(\tau)$, we have $\langle e^*, d^* \rangle \in \text{nco}_{\mu_j \rightarrow \mu_i}(\tau)$. Further,

\[
e = \tilde{e}_{\text{co}}(u, \pi, x) \\
= e_{\text{co}}(u, \pi_1, x) + \tilde{e}(\pi_2 \ldots |\pi|) \\
= 0 + \tilde{e}(\pi_2 \ldots |\pi|) \\
= e^*
\]

and $d = \tilde{d}_{\text{co}}(u, \pi, x) \geq \tilde{d}(\pi_2 \ldots |\pi|) = d^*$. It follows that $\langle e^*, d^* \rangle \trianglerighteq \langle e, d \rangle$.

Case 3 ($e(u) - e(\pi_1) < x < e(u)$):
Again, let $\langle e^*, d^* \rangle = \text{pair}_{\text{co}}(u, \pi, e(u))$. Now,

\[
e = \tilde{e}_{\text{co}}(u, \pi, x) \\
= e_{\text{co}}(u, \pi_1, x) + \tilde{e}(\pi_2 \ldots |\pi|) \\
= e_{\text{co}}(u, \pi_1, e(u)) - e(u) + x + \tilde{e}(\pi_2 \ldots |\pi|) \\
= \tilde{e}_{\text{co}}(u, \pi, e(u)) - e(u) + x \\
= e^* - e(u) + x.
\]

Similarly,

\[
d = \tilde{d}_{\text{co}}(u, \pi, x) \\
\geq \tilde{d}_{\text{co}}(u, \pi, e(u)) - e(u) + x \\
= d^* - e(u) + x.
\]

It follows that $e^* \geq e$ and $e^* - e = e(u) - x \geq d^* - d$, and therefore that $\langle e^*, d^* \rangle \trianglerighteq \langle e, d \rangle$.

\[\blacksquare\]
We can now prove Lemma 3.4.

**Proof of Lemma 3.4** From Eq. (12) and (14) we know that \( \text{tdbf}_{\mu_j \rightarrow \mu_i}^*(\tau, \ell) \) is defined by a subset of the set of demand pairs defining \( \text{tdbf}_{\mu_j \rightarrow \mu_i}^{\text{ub}}(\tau, \ell) \). It follows directly that \( \text{tdbf}_{\mu_j \rightarrow \mu_i}^*(\tau, \ell) \leq \text{tdbf}_{\mu_j \rightarrow \mu_i}^{\text{ub}}(\tau, \ell) \), and the \( \Leftarrow \) direction of the lemma holds.

We instead consider the \( \Rightarrow \) direction. From Eq. (12) it is clear that if there exists an \( \ell_1 \geq 0 \) such that \( \sum_{\tau \in T} \text{tdbf}_{\mu_j \rightarrow \mu_i}^{\text{ub}}(\tau, \ell_1) > \ell_1 \), then for each \( \tau \in T \) there must exist demand pairs \( \langle e_\tau, d_\tau \rangle \in \text{nco}_{\mu_j \rightarrow \mu_i}(\tau) \cup \text{co}_{\mu_j \rightarrow \mu_i}(\tau) \) such that

\[
\sum_{\tau \in T} e_\tau > \ell_1 \quad \text{and} \quad \max_{\tau \in T}(d_\tau) \leq \ell_1.
\]  

(17)

From Lemma C.1 we know that for each of the demand pairs \( \langle e_\tau, d_\tau \rangle \) there exists some demand pair \( \langle e_\tau^*, d_\tau^* \rangle \in \text{nco}_{\mu_j \rightarrow \mu_i}(\tau) \cup \text{co}_{\mu_j \rightarrow \mu_i}(\tau) \) such that \( \langle e_\tau^*, d_\tau^* \rangle \geq \langle e_\tau, d_\tau \rangle \). By Definition C.1 we have

\[
\sum_{\tau \in T} e_\tau^* - \sum_{\tau \in T} e_\tau \geq \max_{\tau \in T}(d_\tau^*) - \max_{\tau \in T}(d_\tau).
\]  

(18)

From Eq. (17) and (18) it follows that

\[
\sum_{\tau \in T} e_\tau^* > \max_{\tau \in T}(d_\tau^*).
\]  

(19)

Let \( \ell_2 = \max_{\tau \in T}(d_\tau^*) \). From the existence of the demand pairs \( \langle e_\tau^*, d_\tau^* \rangle \) and Eq. (14) and (19) we know that

\[
\sum_{\tau \in T} \text{tdbf}_{\mu_j \rightarrow \mu_i}^*(\tau, \ell_2) \geq \sum_{\tau \in T} e_\tau^* > \ell_2.
\]  

\[\blacksquare\]
### Appendix D  Table of Notations Used for the Analysis

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau \in T$</td>
<td>An MS-DRT task $\tau$ in task set $T$</td>
</tr>
<tr>
<td>$\mu_i \in M(T)$</td>
<td>A mode $\mu_i$ in the set of modes of $T$</td>
</tr>
<tr>
<td>$V(\tau)$</td>
<td>Job types (vertices) of task $\tau$</td>
</tr>
<tr>
<td>$E_{cf}(\tau)$</td>
<td>Control-flow edges of task $\tau$</td>
</tr>
<tr>
<td>$E_{ms}(\tau)$</td>
<td>Mode-switch edges of task $\tau$</td>
</tr>
<tr>
<td>$e(v), d(v), \mu(v)$</td>
<td>Execution time, relative deadline and mode of job type $v$</td>
</tr>
<tr>
<td>$G(T)$</td>
<td>Mode structure of task set $T$</td>
</tr>
<tr>
<td>$\text{pred}_{G(T)}(\mu_i)$</td>
<td>Modes that can precede $\mu_i$ in $G(T)$</td>
</tr>
<tr>
<td>$\text{DRT}_{\mu_i}(\tau)$</td>
<td>The subgraph in $(V(\tau), E_{cf}(\tau))$ corresponding to mode $\mu_i$</td>
</tr>
<tr>
<td>$\Pi_{\mu_i}(\tau)$</td>
<td>Set of finite paths through $\text{DRT}_{\mu_i}(\tau)$</td>
</tr>
<tr>
<td>$\pi_n$</td>
<td>The $n$-th vertex on path $\pi$</td>
</tr>
<tr>
<td>$</td>
<td>\pi</td>
</tr>
<tr>
<td>$\pi_{n\cdots m}$</td>
<td>Subpath of $\pi$ between the $n$-th and $m$-th vertices (inclusive)</td>
</tr>
<tr>
<td>$\bar{e}(\pi)$</td>
<td>Cumulative execution time of job types on path $\pi$</td>
</tr>
<tr>
<td>$\bar{d}(\pi)$</td>
<td>Smallest interval size that fits jobs from all vertices on path $\pi$</td>
</tr>
<tr>
<td>$E_{\mu_j \rightarrow \mu_i}(\tau)$</td>
<td>Mode-switch edges from $\mu_j$ to $\mu_i$ in task $\tau$</td>
</tr>
<tr>
<td>$\text{first}_{\mu_j \rightarrow \mu_i}(\tau)$</td>
<td>Vertices in $V(\tau)$ that can directly follow a switch from $\mu_j$ to $\mu_i$</td>
</tr>
<tr>
<td>$e_{co}(u, v, x)$</td>
<td>Max execution left of carry-over job at $(u, v) \in E_{ms}(\tau)$, given $x$</td>
</tr>
<tr>
<td>$d_{co}(u, v, x)$</td>
<td>Remaining scheduling window size of carry-over job, given $x$</td>
</tr>
<tr>
<td>$p_{co}(u, v, w, x)$</td>
<td>Min delay from mode switch to edge $(v, w) \in E_{cf}(\tau)$, given $x$</td>
</tr>
<tr>
<td>$e_{co}(u, \pi, x)$</td>
<td>As $e_{co}(u, v, x)$, generalized to path $\pi$</td>
</tr>
<tr>
<td>$d_{co}(u, \pi, x)$</td>
<td>As $d_{co}(u, v, x)$, generalized to path $\pi$</td>
</tr>
<tr>
<td>$\text{nco}_{\mu_j \rightarrow \mu_i}(\tau)$</td>
<td>Set of demand pairs for paths not starting with a carry-over job</td>
</tr>
<tr>
<td>$\text{co}_{\mu_j \rightarrow \mu_i}(\tau)$</td>
<td>Set of demand pairs for paths starting with a carry-over job</td>
</tr>
<tr>
<td>$\text{co}^*_{\mu_j \rightarrow \mu_i}(\tau)$</td>
<td>Subset of $\text{co}_{\mu_j \rightarrow \mu_i}(\tau)$ of “critical” demand pairs</td>
</tr>
<tr>
<td>$\text{idbf}_{\mu_i}(T, \ell)$</td>
<td>Exact internal demand bound function of $T$</td>
</tr>
<tr>
<td>$\text{tdbf}_{\mu_j \rightarrow \mu_i}(T, \ell)$</td>
<td>Exact transitional demand bound function of $T$</td>
</tr>
<tr>
<td>$\text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell)$</td>
<td>Over-approximate transitional demand bound function of $\tau$</td>
</tr>
<tr>
<td>$\text{idbf}^*_{\mu_i}(\tau, \ell)$</td>
<td>Exact internal demand bound function of $\tau$</td>
</tr>
<tr>
<td>$\text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell)$</td>
<td>Simplified version of $\text{tdbf}^*_{\mu_j \rightarrow \mu_i}(T, \ell)$, preserving schedulability</td>
</tr>
<tr>
<td>$\text{DRT}_{\mu_j \rightarrow \mu_i}(\tau)$</td>
<td>DRT task with demand bound function equal to $\text{tdbf}^*_{\mu_j \rightarrow \mu_i}(\tau, \ell)$</td>
</tr>
<tr>
<td>$\text{S}_{p}(T, \mu_i)$</td>
<td>Internal schedulability of $T$ in $\mu_i$, equivalent to $S^*_{\text{EDF}}(T, \mu_i)$</td>
</tr>
<tr>
<td>$\text{S}_{p}(T, \mu_j \rightarrow \mu_i)$</td>
<td>Transitional schedulability of $\mu_i$ when reached from $\mu_j$</td>
</tr>
<tr>
<td>$\text{S}^*_{\text{EDF}}(T, \mu_j \rightarrow \mu_i)$</td>
<td>Implies $\text{S}_{p}(T, \mu_j \rightarrow \mu_i)$ when $\mu_j$ is schedulable</td>
</tr>
</tbody>
</table>
References


Paper III
An Optimal Resource Sharing Protocol for Generalized Multiframe Tasks

Pontus Ekberg, Nan Guan, Martin Stigge and Wang Yi

Abstract

Many different task models of varying expressiveness have been proposed to capture the timing constraints of real-time workloads. However, techniques for optimal scheduling and exact feasibility analysis of systems with mutually-exclusive shared resources have been available only for relatively simple task models, such as the sporadic task model. We consider sharing of mutually-exclusive resources in task models that are more expressive in terms of the job-release patterns that can be modeled. We propose a new scheduling algorithm and show that it is optimal for scheduling generalized multiframe tasks with shared resources on uniprocessor systems. We also present an efficient feasibility test for such systems, and show that it is both sufficient and necessary.

1 Introduction

Processes in real-time systems often compete for shared resources, such as peripheral devices or global data structures that must be accessed in a mutually exclusive manner. To avoid deadlocks and low processor utilization, we need scheduling algorithms that handle the resource sharing.

Well-established solutions to the resource sharing problem exist in the context of sporadic [8] task sets. However, these existing solutions are generally either inapplicable or suboptimal for more flexible task models. One generalization of the sporadic task model is provided by the generalized multiframe (GMF) task model [3]. With GMF we can model tasks that cycle through a set of different behaviors, and can more precisely represent many systems. Baruah et al. [3] have shown how to efficiently decide feasibility for this model when shared resources are not considered.

The flexible structure of GMF tasks, in combination with shared resources, is the source of difficulty in finding an optimal scheduling strategy. To make optimal scheduling decisions at run-time, we must be aware of the tasks’ structures and which behaviors they may display in the near future.

The goal of this work is to show how to analyze and schedule GMF task sets with shared resources. We introduce an efficient technique, which takes the tasks’ structures into account, to predict possible resource contention at run-
time and thereby determine the urgency of unlocking currently used resources. Based on this technique we propose a new scheduling algorithm and show that it is well suited for scheduling such workloads. The main contributions include:

- We propose the resource deadline protocol (RDP) for handling shared resources, and combine it with earliest deadline first (EDF) to form the EDF+RDP scheduling algorithm. We prove that EDF+RDP has the following properties:
  - It is optimal for scheduling GMF task sets with shared resources, in the sense that it successfully schedules all feasible task sets.
  - It is deadlock-free, and it enables efficient implementations because there is at most one preemption per job release and all jobs in the system can share a common run-time stack.

- We derive a sufficient and necessary feasibility test for GMF task sets with shared resources. This test is in the same complexity class as the known feasibility test for GMF task sets without resources [3], i.e., pseudo-polynomial for bounded-utilization task sets.

2 Preliminaries

The results of this paper are presented in the context of the GMF task model [3], which we describe in Section 2.1. We extend GMF with the ability to model shared resources in Section 2.2.

2.1 The Generalized Multiframe Task Model

The GMF task model is a generalization of the well-known sporadic [8] and multiframe [9] task models. Like a sporadic task, a GMF task releases a sequence of jobs. However, the jobs released by a GMF task do not all need to have the same parameters (e.g., execution time and deadline). Instead, a GMF task cycles through a sequence of job types, which specify the parameters of the jobs that are released.

A natural way of representing a GMF task is to use a directed cycle graph, where the vertices represent the job types, and the arcs specify the order in which jobs are released (as well as the minimum delay between consecutive job releases). An example GMF task is depicted in Figure 1. Formally, a GMF task set \( \tau \) is defined as follows:

- Each task \( T \in \tau \) is a directed cycle graph, with vertices \( V(T) \) and arcs \( A(T) \).

\[ \text{The notation used for GMF tasks in this paper is different from, though equivalent to, the notation used in [3].} \]
• Each vertex $v \in V(T)$ is called a job type and is labeled with a pair $\langle E(v), D(v) \rangle$. For each job that is of type $v$, $E(v) \in \mathbb{N}_+$ is an upper bound on its required execution time, and $D(v) \in \mathbb{N}_+$ is its relative deadline.

• Each arc $(u,v) \in A(T)$ is labeled with a minimum inter-release separation time $P(u,v) \in \mathbb{N}$.

• One vertex $v_0 \in V(T)$, denoted $S(T)$, is called the start vertex of $T$.

We denote the unique successor of vertex $u$ with $\text{succ}(u)$, and note that $\text{succ}(u) = v$ if and only if $(u,v) \in A(T)$.

When the system is running, each task $T$ releases a possibly infinite sequence of jobs $[J_0, J_1, J_2, \ldots]$, where each job corresponds to one of $T$'s job types. Intuitively, a job sequence is produced by traversing the graph of $T$, starting at vertex $S(T)$. Every time a vertex is visited, a job of the corresponding job type is released. Before the next vertex can be visited, the task must wait for at least the minimum inter-release separation time that is labeled on the arc leading there.

Each job $J_i$ in a job sequence is specified by a triple $(r(J_i), e(J_i), d(J_i)) \in \mathbb{R}^3$, where $r(J_i)$ is the job’s absolute release time, $d(J_i)$ its absolute deadline and $e(J_i)$ its execution time requirement. A job sequence is said to be generated by task $T$ if and only if there is a path $[v_0, v_1, v_2, \ldots]$ through $T$ such that the following hold for all $i \geq 0$:

1. $v_0 = S(T)$,
2. $r(J_{i+1}) \geq r(J_i) + P(v_i, v_{i+1})$,
3. $e(J_i) \leq E(v_i)$,
4. $d(J_i) = r(J_i) + D(v_i)$.

A job sequence is generated by a task set $\tau$ if and only if it is an interleaving of job sequences generated by the tasks $T \in \tau$. 

Figure 1. An example GMF task containing five job types.
We assume that the tasks satisfy the \textit{localized Monotonic Absolute Deadlines (l-MAD)} property described in [3]. That is, we assume $D(u) \leq P(u,v) + D(v)$ for all arcs $(u,v)$. This property guarantees that all jobs released by the same task have their (absolute) deadlines ordered in the same order as their release times.

### 2.2 Modeling Shared Resources

In the plain GMF model described above, all jobs are completely independent; there is no way to model contention between jobs for shared resources. Here we extend the GMF model to include non-preemptable shared resources. The extended model allows us to express which resources may be used by jobs of each job type, and for how long. We refer to the extended model as the GMF-R task model.

When a job is granted access to a resource, we say that it \textit{locks} the resource, and then \textit{holds} it for some time before finally \textit{unlocking} it. If a resource is already held by some job, it cannot be locked again until it has been unlocked by the job holding it. A job trying to lock an already locked resource is said to be \textit{blocked} on that resource, and cannot continue to execute until the resource has been unlocked. Note that a job may be preempted while holding a resource, but no other job may use that resource until it is unlocked.

Each job type has a \textit{worst-case access duration} to each resource. Each time a particular resource is locked by a job, that job will execute for at most this duration before unlocking it again. We do not assume any a priori knowledge about exactly when a job locks a resource. We only assume knowledge about which resources it may lock, and for how long it may execute while holding them in the worst case.

Formally, a GMF-R task set is a triple $(\tau, \rho, \alpha)$, such that

- $\tau$ is a GMF task set,
- $\rho$ is a set of resources,
- $\alpha : V(\tau) \times \rho \rightarrow \mathbb{N} \cup \{\bot\}$ is a function mapping job types and resources to their worst-case access durations,

where $V(\tau) = \bigcup_{T \in \tau} V(T)$ is the set of all job types in $\tau$.

The worst-case access duration of jobs of type $v$ to resource $R \in \rho$ is given by $\alpha(v,R)$. If $\alpha(v,R) = \bot$, then jobs of type $v$ do not use resource $R$.\footnote{Note that $\alpha(v,R) = 0$ is useful to express that jobs of type $v$ can be forbidden to execute while some other job holds $R$, but do not hold $R$ themselves. This can be used to model non-preemptable sections in jobs.} Otherwise, $\alpha(v,R) \leq E(v)$ is assumed. We let $\alpha^\text{max}(T,R)$ denote the maximum access duration to resource $R$ by any job type in task $T$. Figure 2 shows an example GMF-R task set with two GMF tasks and two shared resources (only $\alpha(v,R) \neq \bot$ are shown).

\footnote{Note that $\alpha(v,R) = 0$ is useful to express that jobs of type $v$ can be forbidden to execute while some other job holds $R$, but do not hold $R$ themselves. This can be used to model non-preemptable sections in jobs.}
A single job may use several different resources, possibly at the same time, but resource accesses must be properly nested. That is, if a job locks resource $R_1$ and afterwards locks $R_2$ before unlocking $R_1$, it must unlock $R_2$ before unlocking $R_1$. A job may lock the same resource $R$ several times during its execution, but each time executing with it held for at most its worst-case access duration to $R$. A job must unlock all resources that it holds before finishing execution. Any sequence of locking and unlocking events that follows the above rules is called a valid access pattern.

![Figure 2. A GMF-R task set with two tasks and shared resources $R_1$ and $R_2$.](image)

2.3 Scheduling Algorithms and Feasibility

For a job sequence to be successfully scheduled, all jobs must finish their execution before their deadlines. That is, each job $J$ in the sequence must be exclusively executed for $e(J)$ time units (not necessarily continuously) between $r(J)$ and $d(J)$. A job is said to be active during the time interval between its release time and the time point where it finishes execution.

A scheduling algorithm decides at each time point which active, non-blocked job (if any) to execute. A scheduling algorithm can know the current system state and how jobs have been released in the past. It does not know what will happen in the future, other than what is specified by the task model.

**Definition 2.1** (Feasibility and Optimality). A GMF-R task set $(\tau, \rho, \alpha)$ is feasible if and only if there exists some scheduling algorithm that can successfully schedule all job sequences generated by $\tau$, for all valid access patterns to the resources in $\rho$ by jobs in the sequence. A scheduling algorithm is optimal if and only if it can successfully schedule all feasible task sets. □
2.4 A Motivating Example

Here we present a simple example task set, showed in Figure 3, that illustrates the difficulties in scheduling GMF-R task sets in an optimal way. This task set is feasible, which can be confirmed with the feasibility test presented later in Section 5, but it is not schedulable by any previous scheduling algorithm that we know of. To see why, consider a scenario where a job $J$ from $T_1$ is released at time $t$, and locks resource $R_1$. At $t+1$, a job $J'$ from $T_2$ is released. We now have to decide whether $J'$ should preempt $J$. If $J'$ does preempt $J$, locks $R_2$, and immediately afterwards $T_3$ releases a job of type $w_0$, then $w_0$ will be blocked for up to 3 time units waiting for $R_1$ and $R_2$ to be unlocked, missing its deadline. If instead $J'$ does not preempt $J$, and when $J$ is finished $T_3$ releases a job of type $w_1$, then one of the remaining two jobs must miss its deadline, because there is only 4 time units available to finish 5 time units of work.

In fact, to make a safe decision about whether $J'$ should preempt $J$, we must know the state of $T_3$, and analyze its possible behaviors in the near future. The ability to do so is a key mechanism in the optimal scheduling algorithm that we present in the following section. It can be noted that the lack of branching in GMF-R tasks allows us to efficiently make the necessary predictions to achieve optimal scheduling. While the scheduling algorithm we propose could be applied to more general task models that include branching, such as the digraph real-time task model [11] extended with resources, its optimality would be lost.

![Figure 3. A feasible GMF-R task set.](image)

3 The Resource Deadline Protocol

The resource deadline protocol (RDP) is a resource sharing protocol, designed to extend the earliest deadline first (EDF) scheduling algorithm to handle shared
resources. We call the resulting scheduling algorithm EDF+RDP. We will show that EDF+RDP is an optimal scheduling algorithm for GMF-R task sets.

EDF+RDP uses what we call virtual deadlines to schedule jobs. It schedules jobs in a similar way to EDF, but uses virtual deadlines instead of absolute deadlines for scheduling decisions. That is, at each time point, EDF+RDP chooses to run the job with the earliest virtual deadline. It is then up to the RDP part of EDF+RDP to assign virtual deadlines to jobs in a way that guarantees the desired properties. It does this by potentially lowering the virtual deadlines (and thereby increasing the priorities) of jobs that are currently holding resources. The virtual deadline of a job therefore represents not only the urgency of the job itself, but also the urgency of releasing the resources that the job is currently holding. To assign virtual deadlines in an optimal way, we must be able to determine how urgent it is that a certain resource becomes unlocked. We capture this urgency by introducing the concept of a resource deadline, which is described in the following section.

3.1 Resource Deadlines

RDP relies on the idea that we can predict, at any time, exactly the earliest future time point where a not-yet-released job can have a deadline. In particular, we are interested in the deadlines of future jobs that may need some resource $R$. The earliest possible such deadline we call the resource deadline of $R$.

**Definition 3.1** (Resource deadline). The resource deadline $\Delta(R, t)$ of resource $R$ at time point $t$ is exactly the earliest time point when some job that is not yet released at $t$ and that may need $R$ can potentially have a deadline, without violating the semantics of the task model.

In other words, at time $t$, let $[J, J', \ldots, J''']$ be the job sequence that has been released by a task set $(\tau, \rho, \alpha)$ so far (i.e., only containing jobs with release times no larger than $t$). Then $\Delta(R, t)$ is the smallest value such that the following is satisfied, for some potential future job $J'''$:

1. Some $[J, J', \ldots, J'', \ldots, J''']$ is generated by $\tau$,
2. $J'''$ may use $R$,
3. $\Delta(R, t) = d(J''')$,
4. $r(J''') \geq t$.

Note that no future job that uses $R$ actually has to get a deadline at $\Delta(R, t)$, as long as it is possible, given the task model and the system state at time $t$. We will show how resource deadlines can be efficiently computed at run-time in Section 7.

**Example 3.1.** Consider the task system in Figure 2 and the following run-time scenario, illustrated in Figure 4. At time 115, we want to know the resource deadline $\Delta(R_1, 115)$. The latest job released by task $T_1$ was of type $v_3$ at time
111, and the latest job released by task $T_2$ was of type $u_1$ at time 102. We can see that the next job of $T_1$ that may need $R_1$ is of type $v_0$. The earliest possible deadline of the next job of type $v_0$ is at $111 + 5 + 12 + 9 = 137$. Similarly, the next job of $T_2$ that may need $R_1$ is of type $u_2$, and can have a deadline earliest at time $102 + 20 + 35 = 157$. The earliest possible time when some future job that needs $R_1$ may have a deadline is therefore at time 137, and $\Delta(R_1, 115) = 137$.

Figure 4. At time 115 we want to know the resource deadline for $R_1$. The solid arrows indicate release times and deadlines of the latest jobs from $T_1$ and $T_2$ (of types $v_3$ and $u_1$, respectively). The dotted arrows indicate the earliest possible release times and deadlines of future jobs. We can see that the earliest possible deadline of a future job that uses $R_1$ is at time 137.

3.2 The EDF+RDP Scheduling Algorithm

In EDF+RDP we use virtual deadlines to represent the urgency of executing jobs. The urgency of executing a job depends not only on the job itself (i.e., its absolute deadline), but also on whether the resources that it holds might be needed by some other job. We introduced resource deadlines to capture this latter aspect of the urgency.

By combining these aspects of urgency, we can now present the complete EDF+RDP scheduling algorithm, which is defined by the following four rules:

1. At each time point, EDF+RDP executes the active job $J$ with the earliest virtual deadline $v(J)$. If several jobs share this earliest virtual deadline, then those jobs are executed in first-come, first-served order.

2. When a job $J$ is released, it gets a virtual deadline equal to its absolute deadline:

   \[ v(J) \leftarrow d(J). \]

3. When a job $J$ locks a resource $R$ at time $t$, it gets a virtual deadline based on the resource deadline $\Delta(R, t)$:

   \[ v(J) \leftarrow \min(v(J), \Delta(R, t)). \]
4. When a job unlocks a resource, it regains the virtual deadline that it had before locking that resource.

Note that a virtual deadline only changes when the corresponding job locks or unlocks some resource. In particular, the virtual deadline set in rule 3 depends only on the resource deadline at the time of locking.

4 Properties of EDF+RDP

In this section we will show some of the desired properties of EDF+RDP, namely the deadlock freedom, the bounded number of preemptions, and the ability to share a common stack among all jobs. The optimality of EDF+RDP for GMF-R task sets will be established in Section 5. Note that the properties shown in this section hold for any task model that we can compute resource deadlines for, while the optimality is specific for GMF-R (and models generalized by GMF-R).

First we prove an auxiliary lemma.

**Lemma 4.1 (No direct blocking).** No job will try to lock a resource that is already held by another job, when scheduled by EDF+RDP.

**Proof.** We will show that a job $J$ will never try to lock a resource $R$ that is already held by another job $J'$.

First consider the case where $J'$ already holds $R$ when $J$ is released. We know (by rule 3 of EDF+RDP and the definition of $\Delta$) that when $J'$ locked $R$, the virtual deadline of $J'$ was set so that $v(J') \leq d(J)$. We also know that $v(J) = d(J)$ when $J$ is released (by rule 2). By rule 1, $J$ will not be chosen for execution (and cannot try to lock $R$) as long as $v(J') \leq v(J)$. It is clear from rules 3 and 4 that $v(J') \leq v(J)$ will continue to be the case at least until $J'$ unlocks $R$. Therefore $J$ cannot try to lock $R$ until after $R$ has been unlocked.

Now consider the case where $J'$ locks $R$ after $J$ has been released. In this case, $J'$ must have executed at some time point $t$ after $J$'s release (at the time of locking $R$). Since $J'$ was executed at $t$, it must have been chosen at $t$ by EDF+RDP (according to rule 1) in favor of $J$ for execution. By rules 3 and 4, $v(J')$ cannot be later than it was at time $t$ until after $R$ is unlocked. Therefore, $J'$ will continue to be chosen for execution in favor of $J$ while $R$ is locked.

An interesting observation based on Lemma 4.1 is that a system using EDF+RDP has no need to explicitly enforce mutual exclusion, because the scheduling already provides an implicit guarantee that there will be no conflicts.

From Lemma 4.1 it follows that EDF+RDP will never let the system be idle as long as there is any active job.

**Corollary 4.2 (Work-conserving scheduling).** If there are active jobs in the system, EDF+RDP will execute one of them.
Proof. From rule 1 it is clear that as long as there are active jobs, EDF+RDP will pick one for execution, and from Lemma 4.1 we know that this job can always execute.

We can now see that there can be no deadlocks in a system scheduled by EDF+RDP.

**Theorem 4.3** (Deadlock freedom). If a task set is scheduled by EDF+RDP, there can be no deadlocks.

**Proof.** From Corollary 4.2 we know that the system can never be in a state where there are active jobs and no job executes. It immediately follows that there cannot be any deadlocks in the system.

Before we show that the number of context switches is bounded and all jobs can share the same run-time stack, we establish the following auxiliary lemma.

**Lemma 4.4** (No preemption by older jobs). When a job $J$ has started execution, no other job that was active at $J$’s start time will execute until $J$ is finished.

**Proof.** Let $t_a$ be the start time of $J$, i.e., the time when $J$ was first chosen for execution, and $t_b$ be $J$’s finishing time. We will show that no other job $J'$ that was active at $t_a$ will execute in the interval $[t_a, t_b]$.

Assume for the sake of contradiction that such a $J'$ executes in $[t_a, t_b]$, and let $t \in [t_a, t_b]$ be the earliest time when $J'$ was chosen for execution over $J$. We know that when $J$ started at $t_a$, it was chosen by EDF+RDP according to rule 1 over all other active jobs (including $J'$). If $J'$ is to be chosen over $J$ at time $t$, then either $v(J)$ must have increased (been postponed) or $v(J')$ decreased since $t_a$. By rules 2-4 of EDF+RDP, $v(J)$ can never increase beyond its initial value at $t_a$. Also, $v(J')$ can only decrease if $J'$ first executes (and locks a resource), which it did not do in $[t_a, t]$. This contradicts the assumption that $J'$ was chosen over $J$ for execution in $[t_a, t_b]$.

For any scheduling algorithm it is important to keep the number of preemptions, and thereby the number of context switches, as low as possible. For EDF+RDP, there can be at most one preemption per job release, the same as with EDF.

**Theorem 4.5** (At most one preemption). If a task set is scheduled by EDF+RDP, there will be at most one preemption per job release.

**Proof.** From Lemma 4.4 it follows that a job $J$ only can preempt jobs that were released before $J$. It also follows from Lemma 4.4 that as soon as $J$ has started execution, none of the jobs that were released before $J$ will execute until $J$ has finished. Consequently, the only possibility for $J$ to preempt another job is at the time when $J$ first starts execution.
Similar to the stack resource policy [1], EDF+RDP guarantees that all jobs can share a common run-time stack without conflicts. This can simplify implementations and substantially reduce memory requirements.

**Theorem 4.6 (The stack can be shared).** For any task set, all jobs can share a common stack space without conflicts if they are scheduled by EDF+RDP.

**Proof.** To see that the stack can be shared, it is enough to show that once a job \( J \) starts execution (and is allocated space on the stack), no other job that has already been allocated stack space will execute until \( J \) is finished. This follows immediately from Lemma 4.4. \( \square \)

We also note that for GMF-R task sets, EDF+RDP will never execute a job before all other jobs that were previously released by the same task have finished (due to the \( l \)-MAD property). This fact can be important in implementations where jobs depend on results produced by previous jobs from the same task.

## 5 Feasibility and Optimality

Here we present a sufficient and necessary feasibility test for GMF-R task sets. We will show that EDF+RDP meets all deadlines for task sets that pass the test, from which the optimality of EDF+RDP follows.

### 5.1 A Feasibility Test for GMF-R Task Sets

The test builds on the general feasibility test framework based on demand-bound functions. Demand-bound functions have been used for checking feasibility of GMF task sets without resources [3].

**Definition 5.1 (Demand-bound function).** The demand-bound function, \( dbf(T, \ell) \), gives the maximum total workload of jobs generated by task \( T \) that are both released and have deadlines within any time interval of length \( \ell \).

In other words,

\[
dbf(T, \ell) = \max_{[J_0, J_1, J_2, \ldots] \in \mathcal{J}} \left( \max_{t \in \mathbb{R}} \left( \sum \{ e(J_i) \mid r(J_i) \geq t \land d(J_i) \leq t + \ell \} \right) \right),
\]

where \( \mathcal{J} \) is the set of all job sequences generated by \( T \). Intuitively, \( dbf(T, \ell) \) captures the maximum workload from task \( T \) that must be executed in any time interval of length \( \ell \). Note that \( dbf(T, \ell) \) is a non-decreasing step function that changes only at integer values of \( \ell \).

**Example 5.1.** Consider task \( T_1 \) in Figure 2. The maximum possible workload of jobs from \( T_1 \) that are both released and have deadline within any interval of length 40 is given by \( dbf(T_1, 40) \). An example workload is given by a sequence of three jobs of types \( v_3, v_4 \) and \( v_0 \). It is easy to see that these jobs fit into the
interval. The maximum total workload of the three jobs is 2 + 3 + 2 = 7. In fact, this is the maximum workload of any sequence of jobs from \( T_1 \) that fit into an interval of length 40, and therefore \( \text{dbf}(T_1, 40) = 7 \). □

To analyze feasibility for GMF-R task sets, we introduce the concept of a resource-constrained demand-bound function.

**Definition 5.2** (Resource-constrained \( \text{dbf} \)). The resource-constrained demand-bound function, \( \text{dbf}(T, R, \ell) \), is defined exactly as the \( \text{dbf}(T, \ell) \), but with the extra constraint that at least one of the jobs contributing to the workload is of a type that uses resource \( R \).

The idea behind \( \text{dbf}(T, R, \ell) \) is that it captures the maximum workload from task \( T \) that must be executed in any interval of length \( \ell \), such that some job of \( T \) may have to wait for \( R \) in that interval. If \( \alpha_{\text{max}}(T, R) = \bot \), then \( \text{dbf}(T, R, \ell) = 0 \) for all \( \ell \). In Section 6.1 we will describe how the resource-constrained demand-bound function can be computed in polynomial time, by using the algorithm from [3] for computing the standard demand-bound function.

**Example 5.2.** Consider again task \( T_1 \) from Figure 2. In the previous example we saw that three jobs of types \( v_3, v_4 \), and \( v_0 \) produce the maximum workload in any interval of length 40. However, none of these job types uses resource \( R_2 \), so that job sequence is not considered for \( \text{dbf}(T_1, R_2, 40) \). Instead, the maximum workload of jobs that fit into the interval, such that at least one job uses resource \( R_2 \), comes from a job of type \( v_1 \) followed by a job of type \( v_2 \). This workload is \( 1 + 5 = 6 \), and therefore \( \text{dbf}(T_1, R_2, 40) = 6 \). □

We can now present the new feasibility test.

**Definition 5.3** (Feasibility test). A GMF-R task set \((\tau, \rho, \alpha)\) passes the test if and only if both of the following two conditions hold.

**Condition A:** For all \( \ell \geq 0 \):

\[
\sum_{T \in \tau} \text{dbf}(T, \ell) \leq \ell.
\]

**Condition B:** For all \( \ell \geq 0 \), all resources \( R \in \rho \), and all tasks \( T, T' \in \tau \), such that \( T \neq T' \), \( \alpha_{\text{max}}(T, R) \neq \bot \) and \( \text{dbf}(T', R, \ell) > 0 \):

\[
\alpha_{\text{max}}(T, R) + \text{dbf}(T', R, \ell) + \sum_{T'' \in \tau \setminus \{T, T'\}} \text{dbf}(T'', \ell) \leq \ell.
\]

Intuitively, Condition A captures the case when no job waits for shared resources, and Condition B the case when waiting is involved. The three terms in the left-hand side (LHS) of Condition B represent a task \( T' \) with a job holding a resource \( R \), a task \( T'' \) with a job that needs \( R \), and all other tasks, respectively.
We will establish the complexity of the feasibility test in Section 6. It is pseudo-polynomial for task sets with a utilization bounded by a constant strictly smaller than 1. This is the same complexity as for the known feasibility tests for sporadic task sets [4] and GMF task sets without resources [3], and is considered tractable for this type of problem.

5.2 Necessity, Sufficiency and Optimality

We show that the feasibility test is both necessary and sufficient.

**Theorem 5.1** (Necessity). If a GMF-R task set \((\tau, \rho, \alpha)\) fails the feasibility test, then \((\tau, \rho, \alpha)\) is infeasible.

*Proof.* If Condition A fails for some \(\ell\), then the tasks in \(\tau\) can together require to be executed for more than \(\ell\) time units in some interval of length \(\ell\). Clearly, no scheduler can achieve this.

Consider instead the case where Condition B fails for some combination of \(\ell, R, T\) and \(T'\), and let \(\ell + k\) be the value of the LHS of Condition B. We know that \(k > 0\).

We construct a scenario in which no scheduler can meet all deadlines. First, let the only active job be a job from \(T\) that may use resource \(R\) for \(\alpha_{\text{max}}(T, R)\) time units. Sooner or later, this job must be executed (by any scheduler) or it will miss its deadline. When it is executed, it locks resource \(R\) at some time point \(t - \epsilon\), where \(\epsilon < k\) and \(\epsilon \leq \alpha_{\text{max}}(T, R)\). It will hold \(R\) for as long as possible. At time \(t\), task \(T'\) starts releasing the job sequence that corresponds to the value of \(dbf(T', R, \ell)\). By the definition of the resource-constrained demand-bound function, \(dbf(T', R, \ell)\) time units of work from \(T'\) must then be executed in the interval \([t, t + \ell]\) to avoid a deadline miss. In addition, at least one job from \(T'\) has to wait for \(R\) to be unlocked before it can finish, so the job from \(T\) must also execute for another \(\alpha_{\text{max}}(T, R) - \epsilon\) time units in that interval in order to unlock \(R\). Finally, all other tasks \(T''\) release the job sequences that correspond to \(dbf(T'', \ell)\), starting at time \(t\). The total workload from all tasks that must be executed in \([t, t + \ell]\) sums up to

\[
\alpha_{\text{max}}(T, R) + dbf(T', R, \ell) + \sum_{T'' \in \tau \setminus \{T, T'\}} dbf(T'', \ell) - \epsilon.
\]

In other words, \(\ell + k - \epsilon\) time units of work must be executed in an interval of length \(\ell\), which no scheduler can do.

Before we can prove the sufficiency of the feasibility test, we need to describe the concept of a busy period, which is defined whenever a deadline is missed.

**Definition 5.4** (Busy period). Let \(t_d\) be the first time instant at which any job misses its deadline when scheduled by EDF+RDP. The busy period is the...
longest time interval \([t_s, t_d]\), such that in the whole of \([t_s, t_d]\) there are active jobs with absolute (non-virtual) deadlines latest at \(t_d\). □

The busy period captures the critical time interval leading up to a deadline miss. If a job has an absolute deadline latest at \(t_d\) and is active sometime in the busy period, we call it a pressing job. Note that a pressing job, by definition, is both released and has its deadline inside the busy period. A job that executes in the busy period with an absolute deadline after \(t_d\) is called a blocking job.

In the following lemma we show some properties of the busy period and of blocking jobs.

**Lemma 5.2** (Contention). If EDF+RDP schedules a task set and has a first deadline miss at \(t_d\), and \(J\) is a blocking job executing in the busy period \([t_s, t_d]\), then there exists an earliest time point \(t_l \leq t_s\) such the following hold:

(i) \(J\) locked a resource \(R\) at \(t_l\) with \(\Delta(R, t_l) \leq t_d\), and still held \(R\) at \(t_s\).
(ii) \(J\) only executes in \([t_l, t_d]\) while holding such an \(R\).
(iii) \(J\) is the only blocking job.
(iv) All other jobs \(J' \neq J\) that execute in \([t_l, t_d]\) have both \(r(J') \geq t_l\) and \(d(J') \leq t_d\).

**Proof.** We prove the four properties in turn. Figure 5 serves as an illustration.

Property (i): By rules 2, 3 and 4 of EDF+RDP, we know that \(v(J') \leq d(J')\), for all jobs \(J'\). By the definition of the busy period, there must then exist at least one active, pressing job with virtual deadline latest at \(t_d\) at all time points in \([t_s, t_d]\). Hence, if a blocking job \(J\) executes in the busy period, we know by rule 1 of EDF+RDP that \(v(J) \leq t_d\) while it executes (or some pressing job would execute instead).

A blocking job \(J\) can only have received \(v(J) \leq t_d\) by locking some resource \(R\) at some time \(t_l\), where \(\Delta(R, t_l) \leq t_d\). In order to lock a resource it must execute, and therefore it must have locked one such resource at a time point \(t_l \leq t_s\). Furthermore, \(J\) must still hold one such resource at \(t_s\) to execute in the busy period, since after unlocking a resource \(J\) regains the virtual deadline that it had before locking it. This proves property (i).

Property (ii): Let \(R\) be the first resource that \(J\) locks, such that property (i) is satisfied. Immediately before \(J\) locked \(R\) we had \(v(J) > t_d\). This follows from the fact that resource accesses are properly nested, and the assumption that \(R\) is the first resource locked by \(J\) such that property (i) is satisfied. If \(v(J) \leq t_d\) when \(J\) locked \(R\), then \(J\) already held a resource satisfying property (i), contradicting our assumption. As soon as \(R\) is unlocked we therefore have \(v(J) > t_d\) by property 4 of EDF+RDP, which means that \(J\) will not execute any more in the busy period. This proves property (ii).
Property (iii): We assume, without loss of generality, that \( J \) is the first blocking job to lock a resource \( R \) that satisfies property (i). We know then that \( J \) holds \( R \), and consequently that \( v(J) \leq t_d \) in the interval \([t_l, t_s]\). By rule 1 of EDF+RDP, no job \( J' \) with \( v(J') > t_d \) can execute in \([t_l, t_s]\) or in the busy period \([t_s, t_d]\). In particular, no such job \( J' \) can lock a resource and get \( v(J') \leq t_d \) after \( t_l \). Since \( J \) was the first blocking job to lock a resource that allows it to execute in the busy period, and no other job can lock such a resource after \( J \), we know that at most one blocking job executes in the busy period. This proves property (iii).

Property (iv): Now consider the other jobs \( J' \neq J \) that execute in \([t_l, t_d]\), where \( J \) is the blocking job that locked the first resource \( R \) satisfying property (i) at \( t_l \). We know that there were no active jobs \( J' \) with \( v(J') \leq t_d \) at \( t_l \) (because \( J \) executes at \( t_l \) with \( v(J) > t_d \)). We have also seen that only jobs \( J' \) with \( v(J') \leq t_d \) can execute in \([t_l, t_d]\). Since there were no such jobs active at \( t_l \), only jobs released after \( t_l \) will execute in that interval. Also, the jobs \( J' \) must have \( v(J') \leq t_d \) already when they start execution (or they would not start before \( t_d \) at all). Since \( v(J') = d(J') \) when \( J' \) starts, we know that \( d(J') \leq t_d \). This proves property (iv).

![Figure 5](image.png)

*Figure 5.* If a blocking job \( J \) executes in the busy period, it must have locked some resource \( R \) in a way similar to the above, as is claimed by property (i) of Lemma 5.2.

With the above lemma we can show that EDF+RDP can miss a deadline only for task sets that fail the feasibility test.

**Theorem 5.3 ( Sufficiency ).** If a GMF-R task set \((\tau, \rho, \alpha)\) passes the feasibility test, then EDF+RDP successfully schedules \((\tau, \rho, \alpha)\).

**Proof.** We prove the contrapositive: if EDF+RDP misses a deadline for a GMF-R task set \((\tau, \rho, \alpha)\), then \((\tau, \rho, \alpha)\) fails the feasibility test. Let \([t_s, t_d]\) be the busy period, and let \( J \) be the job that misses its deadline at \( t_d \). From Lemma 5.2 we know that EDF+RDP executes at most one blocking job in the busy period. We separately consider the cases in which there is no blocking job and exactly one blocking job, respectively.

**Only pressing jobs execute in the busy period:** Let \( \ell = t_d - t_s \) be the length of the busy period. By the definition of the busy period we know that \( J \) is a...
pressing job. Since EDF+RDP is work conserving (by Corollary 4.2) and there are active jobs during the entire busy period, it must be the case that EDF+RDP executes some jobs during the whole of \([t_s, t_d]\). We know that \(J\) missed its deadline at \(t_d\), so the total time spent executing other jobs during \([t_s, t_d]\) must be strictly more than \(\ell - e(J)\).

Since all pressing jobs are both released and have their deadlines in the busy period, we know that the total workload executed in \([t_s, t_d]\) is bounded from above by \(\sum_{T \in \tau} \text{dbf}(T, \ell)\). The workload of jobs \(J' \neq J\) executing in \([t_s, t_d]\) is bounded by \(\sum_{T \in \tau} \text{dbf}(T, \ell) - e(J)\). It must be the case that \(\sum_{T \in \tau} \text{dbf}(T, \ell) - e(J) > \ell - e(J)\), and Condition A cannot hold.

Exactly one blocking job executes in the busy period: Let \(R\) be the first resource locked by the blocking job that satisfies property (i) of Lemma 5.2, and let \(t_l\) be the time when it was locked. Let \(\ell = t_d - t_l\).

We know that the blocking job is released latest at \(t_l\) and has deadline after \(t_d\). Since the tasks satisfy the \(l\)-MAD property (described in Section 2.1), no job \(J'\) that is from the same task \(T\) as the blocking job can have both \(r(J') \geq t_l\) and \(d(J') \leq t_d\). The blocking job is therefore the only job from \(T\) that can execute in \([t_l, t_d]\) by property (iv) of Lemma 5.2. We can see then, by property (iv) of Lemma 5.2, that the executed workload of jobs from \(T\) in \([t_l, t_d]\) is bounded by \(\alpha_{\text{max}}(T, R)\).

Since \(\Delta(R, t_l) \leq t_d\), there must be some other task \(T'\) that can release a sequence of jobs from \(t_l\) on, in which at least one job needs \(R\) and has a deadline before \(t_d\). By property (iv) of Lemma 5.2, only jobs with both release time and deadline in \([t_l, t_d]\) execute in that interval, so the workload of task \(T'\) in \([t_l, t_d]\) is bounded by \(\text{dbf}(T', R, \ell)\).

Similarly, the workload of each of the remaining tasks \(T''\) in the interval \([t_l, t_d]\) is bounded by \(\text{dbf}(T'', \ell)\).

The total workload of jobs executing in \([t_l, t_d]\) is thus bounded by

\[
W = \alpha_{\text{max}}(T, R) + \text{dbf}(T', R, \ell) + \sum_{T'' \in \tau \setminus \{T, T'\}} \text{dbf}(T'', \ell).
\]

Note that \(W\) is equal to the LHS of Condition B. We know that EDF+RDP is work conserving and that there are active jobs in the entire interval \([t_l, t_d]\). Since \(J\) missed its deadline, strictly more than \(\ell - e(J)\) time units must therefore have been spent executing other jobs than \(J\) in \([t_l, t_d]\).

The total workload of the other jobs is bounded from above by \(W - e(J)\). Therefore, \(W - e(J) > \ell - e(J)\), and Condition B cannot hold.

We can now establish the optimality of EDF+RDP for GMF-R tasks sets.
Theorem 5.4 (Optimality). EDF+RDP will successfully schedule all feasible GMF-R task sets.

Proof. From Theorems 5.1 and 5.3 we know that the feasibility test is both necessary and sufficient. Further, we know from Theorem 5.3 that EDF+RDP successfully schedules all task sets that pass the feasibility test. The optimality of EDF+RDP follows directly. ■

6 Complexity of the Test

The following two questions must be answered in order to evaluate the feasibility test from Section 5:

1. How do we compute resource-constrained demand-bound functions?
2. For which values of $\ell$ do we need to evaluate Conditions A and B in the feasibility test?

We show how resource-constrained demand-bound functions can be computed in polynomial time in Section 6.1. In Section 6.2 we show that it is enough to evaluate Conditions A and B for pseudo-polynomially many different values for $\ell$, if the utilization of the task set is bounded by a constant strictly smaller than 1.

The feasibility test can therefore be evaluated in pseudo-polynomial time for such bounded-utilization task sets, similarly to feasibility tests for sporadic task sets [4] and GMF task sets without resources [3].

6.1 Computing Resource-Constrained dbfs

Here we describe a polynomial-time algorithm (see Figure 6) for computing resource-constrained demand-bound functions. We make use of the fact that it is already known how to compute the standard demand-bound function in polynomial time [3], by invoking it as a subroutine in the algorithm.

The algorithm in Figure 6 first checks whether any jobs from $T$ uses resource $R$ at all (lines 1-3), otherwise it returns 0. It then checks (lines 5-7) whether the value of $\ell$ is large enough to cycle through the entire graph of $T$ and include each job type at least once. If that is the case, some job type using resource $R$ will for sure be included in the interval, and hence $dbf(T, R, \ell) = dbf(T, \ell)$. We can use the known polynomial-time algorithm [3] to compute $dbf(T, \ell)$.

For smaller $\ell$ we cannot use the algorithm for computing $dbf(T, \ell)$ because we must make sure that some job type that uses $R$ is included. To compute the $dbf(T, R, \ell)$ in this case, we try all possible start vertices (the first job type to be released in the interval of length $\ell$) and walk forward in the graph. For each start vertex, we simply add up as much execution demand as possible in an interval of size $\ell$, and check if we visit at least one job type that uses $R$. Note
1: if $\alpha_{\text{max}}(T, R) = \bot$ then
2: return 0
3: end if
4: if $\ell \geq \sum_{(u,v) \in A(T)} P(u, v) + \max_{v \in V(T)} (D(v))$ then
5: return $\text{dbf}(T, \ell)$
6: end if
7: $e_{\text{max}} \leftarrow 0$
8: for $v \in V(T)$ do
9: $e \leftarrow 0$
10: $t \leftarrow \ell$
11: $r \leftarrow \text{false}$
12: while $t \geq D(v)$ do
13: $e \leftarrow e + E(v)$
14: $t \leftarrow t - P(v, \text{succ}(v))$
15: if $\alpha(v, R) \neq \bot$ then
16: $r \leftarrow \text{true}$
17: end if
18: $v \leftarrow \text{succ}(v)$
19: end while
20: if $(r = \text{true}) \land (e > e_{\text{max}})$ then
21: $e_{\text{max}} \leftarrow e$
22: end if
23: end for
24: return $e_{\text{max}}$

Figure 6. Algorithm for computing $\text{dbf}(T, R, \ell)$.

that the inner while-loop will iterate no more than $O(|V(T)|)$ times (since $\ell$ is small), and the complexity of the algorithm is $O(|V(T)|^2)$ in this case.

6.2 Bounding the Values of $\ell$ in the Test

In the feasibility test, the inequalities in Conditions A and B must hold for all $\ell \geq 0$. The LHS of both inequalities change only at integer-valued $\ell$, so it is enough to check integers $\ell$. Still, there are infinitely many such $\ell$, and we cannot check them all. We show here that it is enough to check the inequalities for $\ell$ smaller than a given bound.

We derive such upper bounds $\ell_{\text{max}}^A$ and $\ell_{\text{max}}^B$ on the values of $\ell$ that must be checked in Conditions A and B, respectively. Both $\ell_{\text{max}}^A$ and $\ell_{\text{max}}^B$ are pseudo-polynomial in the task set representation, if the utilization of the task set is bounded from above by a constant strictly smaller than 1. The total number of different values for $\ell$ that needs to be checked is therefore pseudo-polynomial.
**Definition 6.1** (Utilization). The utilization of a task set is the maximum execution demand that it can create per time unit, asymptotically.

For a GMF task set $\tau$, the utilization $U(\tau)$ is computed as

\[
E^{\text{sum}}(T) = \sum_{v \in V(T)} E(v),
\]
\[
P^{\text{sum}}(T) = \sum_{(u,v) \in A(T)} P(u,v),
\]
\[
U(T) = \frac{E^{\text{sum}}(T)}{P^{\text{sum}}(T)},
\]
\[
U(\tau) = \sum_{T \in \tau} U(T).
\]

We first derive $\ell_A^{\text{max}}$. Consider any sequence of jobs released by a single task $T$ in an interval of length $\ell$. If the job sequence is long enough, some job types will be represented several times in the sequence. We can divide the sequence into one part consisting of full cycles through the graph of $T$, and one part consisting of jobs that are not part of a full cycle. If we make the part with full cycles as large as possible, then at most one job per job type can be in the other part.

The value $\ell \cdot U(T)$ is an upper bound of the workload of jobs in the full cycles (because full cycles generate at most $U(T)$ amount of work per time unit). Also, $E^{\text{sum}}(T)$ is clearly an upper bound of the workload that is not part of a full cycle. Therefore, it must be the case that

\[
E^{\text{sum}}(T) + \ell \cdot U(T) \geq dbf(T, \ell)
\]

and

\[
\sum_{T \in \tau} [E^{\text{sum}}(T) + \ell \cdot U(T)] \geq \sum_{T \in \tau} dbf(T, \ell).
\]

Note that the RHS of the inequality above is equal to the LHS in Condition A. Any $\ell$ that violates the inequality in Condition A must therefore be bounded so that

\[
\sum_{T \in \tau} [E^{\text{sum}}(T) + \ell \cdot U(T)] > \ell.
\]

By rearranging some terms in the above equation we get

\[
\frac{\sum_{T \in \tau} E^{\text{sum}}(T)}{1 - U(\tau)} > \ell,
\]

which gives our bound

\[
\ell_A^{\text{max}} = \frac{\sum_{T \in \tau} E^{\text{sum}}(T)}{1 - U(\tau)}.
\]

The value of $\ell_A^{\text{max}}$ is clearly pseudo-polynomial if $U(\tau)$ is bounded by some constant strictly smaller than one. This bound is in fact equivalent to the bound
derived in [11] for a similar problem concerning the more general digraph real-time task model.

Now consider $\ell_B^{\text{max}}$. One can see that $\ell_B^{\text{max}} = \ell_A^{\text{max}}$ is a valid bound also for Condition B. However, that bound is unnecessarily large for Condition B, so we instead derive another, much smaller $\ell_B^{\text{max}}$. This is not necessary to establish the complexity of the feasibility test, but will substantially speed up implementations.

Let $E_{\text{max}}(T)$ be the largest execution time requirement of all job types in task $T$, and let $D_{\text{max}}(T)$ be the longest (relative) deadline. By definition we have $\alpha_{\text{max}}(T, R) \leq E_{\text{max}}(T)$ and $E_{\text{max}}(T) \leq dbf(T, D_{\text{max}}(T))$. Since $dbf(T, \ell)$ is monotonically non-decreasing in $\ell$, we have for all $\ell \geq D_{\text{max}}(T)$:

$$\alpha_{\text{max}}(T, R) \leq dbf(T, \ell).$$

We also know, by definition, that for all $\ell$:

$$dbf(T, R, \ell) \leq dbf(T, \ell).$$

It follows that the LHS of Condition A is at least as big as the LHS of Condition B for any $\ell \geq \max_{T \in \tau}(D_{\text{max}}(T))$. If there exists such an $\ell$ for which Condition B does not hold, we can be sure that Condition A will not hold for this $\ell$ either. It is therefore not necessary to check these values for $\ell$ also in Condition B, and as the upper bound we can use

$$\ell_B^{\text{max}} = \max_{T \in \tau}(D_{\text{max}}(T)).$$

7 Implementing EDF+RDP

To implement EDF+RDP we must be able to compute resource deadlines efficiently at run-time. Here we show how this can be done for GMF-R task sets.

For any GMF-R task, we can pick two job types from that task and statically calculate the minimum amount of time that must pass between a release of a job of the first type and a deadline of a job of the second type. We can use such pre-computed values to efficiently evaluate resource deadlines at runtime. In particular, for each job type $v$ and resource $R$, we want to pre-compute the minimum amount of time that must pass between a release of $v$ and a deadline of some job (from the same task) that is released no earlier and can use resource $R$. We denote this value $\delta(v, R)$.

Example 7.1. Consider task $T_2$ in Figure 2. After the release of a job of type $u_1$, the next job from $T_2$ that can use $R_1$ is of type $u_2$. Between a release of $u_1$ and the next deadline of $u_2$ there must be at least $20 + 35 = 55$ time units. Hence, $\delta(u_1, R_1) = 55$. □
Formally, \( \delta(v, R) \) is defined as follows. Let \( \text{succ}^i(v) \) denote the \( i \)th successor of \( v \), i.e., \( \text{succ}^0(v) = v \) and \( \text{succ}^i(v) = \text{succ}(\text{succ}^{i-1}(v)) \) if \( i > 0 \). If there are no job types in \( v \)'s task \( T \) that use \( R \), i.e., if \( \alpha^{\text{max}}(T, R) = \perp \), then \( \delta(v, R) = \infty \). Otherwise,

\[
k = \min\{i \geq 0 | \alpha(\text{succ}^i(v), R) \neq \perp\},
\]

\[
\delta(v, R) = \sum_{i=0}^{k-1} P(\text{succ}^i(v), \text{succ}^{i+1}(v)) + D(\text{succ}^k(v)).
\]

Clearly, each \( \delta(v, R) \) can be computed in \( O(|V(T)|) \) time, where \( v \in V(T) \).

Now consider how these pre-computed values can be used to evaluate resource deadlines at run-time. If the next job to be released by a task \( T \) is of type \( v \), then at least \( \delta(v, R) \) time must pass after that release before a deadline of any future job from \( T \) that needs \( R \). To find the earliest possible absolute time point for this deadline, we need to keep track of two things:

1. The type of \( T \)'s next job.
2. The earliest time point where that job can be released.

We keep a vector \( G \) containing the relevant system state, indexed by the tasks \( T \in \tau \). Each entry \( G[T] \) is a pair \((v, t)\), containing the type of \( T \)'s next job and the earliest time point where it could possibly be released without violating the inter-release separation constraints. To keep the vector updated, we simply update one entry per job release. If \( T \) releases a job of type \( v \) at time \( t \):

\[
G[T] \leftarrow (\text{succ}(v), t + P(v, \text{succ}(v))).
\]

To compute a resource deadline \( \Delta(R, t) \) at run-time, we traverse \( G \), and for each entry calculate the earliest possible time point where a job using \( R \) from the corresponding task can have a deadline. If the current time is \( t \) and \( G[T] = (v, t') \), then the earliest time point where the next job from \( T \) can be released is \( \max(t, t') \), and the earliest deadline of a future job from \( T \) that needs \( R \) is \( \max(t, t') + \delta(v, R) \). The resource deadline can therefore be computed as

\[
\Delta(R, t) \leftarrow \min_{(v, t') \in \mathcal{G}} (\max(t, t') + \delta(v, R)),
\]

where \( \mathcal{G} = \{G[T] | T \in \tau \} \).

When the system is first started at time \( t \), the entries of \( G \) are initialized such that \( G[T] \leftarrow (S(T), t) \), where \( S(T) \) is the start vertex of \( T \).

Following the EDF+RDP rules, we need to evaluate a resource deadline only when a resource is being locked. The overhead of computing resource deadlines is therefore \( O(|\tau|) \) per lock operation.

8 Related Work

Sha et al. proposed the classic priority inheritance and priority ceiling protocols [10]. They are mainly used to improve fixed-priority scheduling of sys-
tems with shared resources by effectively bounding the worst-case blocking times.

Baker proposed the stack resource policy (SRP) [1], which works well with dynamic-priority scheduling algorithms. An interesting property of SRP is that it allows all jobs to share a common run-time stack without conflicts, a property which is shared with the algorithm proposed in this paper. Multi-unit resources are also supported by SRP. Baruah [2] later showed that a particular instantiation of SRP combined with EDF is optimal for scheduling sporadic task sets with shared resources. Unfortunately, this optimality does not carry over to GMF-R task sets for the reasons discussed in Section 2.4.

Other relevant work include the dynamic priority ceiling protocol by Chen and Lin [5], and the dynamic deadline modification strategy by Jeffay [7]. Both essentially target a special case of the sporadic task model, where relative deadlines are equal to periods. The scheduling approach taken in this paper resembles Jeffay’s, but targets the more general GMF task model.

The absolute-time ceiling protocol by Guan et al. [6] handles resource sharing with the digraph real-time task model [11], of which GMF is a special case. That protocol does not provide optimal scheduling, and restricts itself to systems without nested resource accesses.

9 Conclusions

We have introduced the EDF+RDP algorithm and a feasibility test for GMF task sets with shared resources. We have shown that EDF+RDP is optimal for this task model, and has a range of other desirable properties. For the feasibility test, we have shown that it is both sufficient and necessary, and runs as efficiently as the known feasibility test that does not consider shared resources. Previously, optimal scheduling algorithms have not been known for flexible task models such as GMF when shared resources are used.

The key to the optimality of EDF+RDP is its ability to predict possible resource contention in future system states, based on the current system state. The GMF task model is well suited for this because the order in which jobs are released by a single task is fixed, even though the jobs may have differing parameters and non-deterministic release times. This can be contrasted to even more general task models, such as the digraph real-time task model [11], which expresses possible job releases with an arbitrary directed graph. The non-deterministic branching inherent to such models makes optimal scheduling very hard to achieve when shared resources are considered. Indeed, while EDF+RDP is applicable to such systems (resource deadlines can be computed in a very similar way), the optimality is lost.

It is interesting to note that the results of this paper can be used in frameworks for hierarchical scheduling and compositional analysis, where several task sets are executed on the same platform but on isolated virtual servers.
The feasibility test described earlier in this paper can easily be extended to this case by replacing the RHS of both Conditions A and B (i.e., the $\ell$) with a supply-bound function $sbf(\ell)$, which lower-bounds the processor supply to the server in any time window of size $\ell$. In such a setting we might have to use another value for the bound $\ell_A^{max}$ though. We can construct a component that contains a GMF task set with (internally) shared resources. If the component is scheduled by EDF+RDP on top of a virtual server, it is guaranteed to meet all deadlines if the extended feasibility test is passed.

References


Uniprocessor Feasibility of Sporadic Tasks with Constrained Deadlines is Strongly coNP-complete

Pontus Ekberg and Wang Yi

Abstract
Deciding the feasibility of a sporadic task system on a preemptive uniprocessor is a central problem in real-time scheduling theory. The computational complexity of this problem has been a long-standing open question. We show that it is coNP-complete in the strong sense, even when deadlines are constrained. This is achieved by means of a pseudo-polynomial transformation from the strongly NP-hard Simultaneous Congruences Problem to the complement of the feasibility problem.

1 Introduction
We let a sporadic task system be defined as a finite multiset $T$ of tasks, where each task is a triple $(e, d, p) \in \mathbb{N}_+^3$, representing its worst-case execution time, relative deadline and minimum inter-arrival separation (or period), respectively.

A sporadic task generates potentially unbounded sequences of jobs. A job is an instance of the task’s workload, characterized by a release time, an execution time and an absolute deadline. A job from task $(e, d, p)$ has execution time not larger than $e$ time units and absolute deadline exactly $d$ time units after its release time. Release times of two consecutive jobs from $(e, d, p)$ are separated by at least $p$ time units. A sporadic task system $T$ generates any interleaving of job sequences that can be generated by each of the tasks $(e, d, p) \in T$.

For a sequence of jobs to be successfully scheduled, every job must be executed for a total duration equal to its execution time, between its release time and its absolute deadline. In the following we assume a preemptive uniprocessor, meaning that only one job can be executed at a time, but a job can be paused and resumed at a later time at no additional cost.

Definition 1.1 (Feasibility). A task system $T$ is feasible if and only if there is some scheduling algorithm that will successfully schedule all job sequences that can be generated by $T$. \[\square\]
A task system $T$ is said to have *implicit deadlines* if $d = p$ for all $(e, d, p) \in T$, and said to have *constrained deadlines* if $d \leq p$ for all $(e, d, p) \in T$. The utilization $U(T)$ of a task system $T$ is defined as

$$U(T) \overset{\text{def}}{=} \sum_{(e,d,p) \in T} \frac{e}{p} \tag{1}$$

Liu and Layland [9] showed that a task system $T$ with implicit deadlines is feasible if and only if $U(T) \leq 1$, but this is not a sufficient condition with constrained deadlines. Dertouzos [4] showed that Earliest Deadline First (EDF) is an optimal scheduling algorithm on preemptive uniprocessors, which means that the terms “feasibility” and “EDF-schedulability” can be used interchangeably here.

**Theorem 1.1** ([4]). A task system $T$ is feasible if and only if it is EDF-schedulable. $\square$

A related workload model is that of (strictly) periodic tasks, where each task is a quadruple $(s, e, d, p) \in \mathbb{N} \times \mathbb{N}_+^3$. The difference to the sporadic task model is that the release times of two consecutive jobs from task $(s, e, d, p)$ must be exactly $p$ time units apart, and the release time of the first job is fixed at time point $s$. A periodic task system is *synchronous* if $s = 0$ for all tasks $(s, e, d, p)$, and *asynchronous* otherwise. It is known [1] that feasibility testing of sporadic tasks is equally hard as that of synchronous periodic tasks, which means that the terms “sporadic” and “synchronous periodic” can be used interchangeably for the results in this paper as well.

**Theorem 1.2** ([1]). A sporadic task system $T$ is feasible if and only if the synchronous periodic task system $\{(0, e, d, p) \mid (e, d, p) \in T\}$ is feasible. $\square$

A classic result is that the feasibility problem for asynchronous periodic task systems with constrained deadlines is strongly $\text{coNP}$-complete [7, 3]. Baruah et al. [3, 1] also developed a pseudo-polynomial time algorithm for the special case of synchronous periodic (or sporadic) task systems with utilization *a priori* bounded by some constant $c < 1$. However, the complexity of the general synchronous case remained open for many years. It was listed as one of five “open problems” in real-time scheduling by Baruah and Pruhs [2]. Shortly thereafter it was partially resolved by Eisenbrand and Rothvoß [5] who showed it to be weakly $\text{coNP}$-hard, but the question remained if it allowed a pseudo-polynomial time solution like the special case with bounded utilization. Eisenbrand and Rothvoß conjectured that it did, but we show that the feasibility problem for synchronous periodic task systems with constrained deadlines is strongly $\text{coNP}$-complete, and thus that it can have no pseudo-polynomial time solution unless $P = \text{NP}$. Figure 1 summarizes the current knowledge.
General case

<table>
<thead>
<tr>
<th>Asynchronous periodic tasks</th>
<th>Utilization bounded by some constant $c$, such that $0 &lt; c &lt; 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>coNP-complete in the strong sense [3]</td>
<td>coNP-complete in the strong sense [3]</td>
</tr>
<tr>
<td>Synchronous periodic tasks (or sporadic)</td>
<td>coNP-complete in the strong sense (from this work)</td>
</tr>
</tbody>
</table>

Figure 1. The current knowledge on the feasibility problem of periodic tasks with constrained deadlines on preemptive uniprocessors.

2 Preliminaries

2.1 The Simultaneous Congruences Problem

The Simultaneous Congruences Problem (SCP) is a number-theoretic decision problem that has been used to establish several complexity results in real-time scheduling theory.

Definition 2.1 (The Simultaneous Congruences Problem). An instance is defined by a multiset $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ and an integer $k$, such that $2 \leq k \leq n$ and $(a_i, b_i) \in \mathbb{N} \times \mathbb{N}_+$ for all $i \in \{1, \ldots, n\}$.

The simultaneous congruences problem asks whether there exists a subset $A' \subseteq A$ of at least $k$ elements and an $x \in \mathbb{N}$, such that

$$x \equiv a_i \pmod{b_i}$$

for all $(a_i, b_i) \in A'$.

In the remainder of this paper we assume, without loss of generality, that $a_i < b_i$ for all $(a_i, b_i) \in A$. □

SCP was first shown to be weakly NP-complete by Leung and Whitehead [8] via a reduction from CLIQUE, and then used by them to show various hardness results concerning fixed-priority scheduling. Leung and Merrill [7] reduced SCP to the complement of the asynchronous periodic feasibility problem on uniprocessors, thus showing that problem to be weakly coNP-hard. Baruah et al. [3] later showed that SCP is in fact strongly NP-complete via an alternative reduction from 3-SAT, which then implied the strong coNP-hardness of the asynchronous periodic feasibility problem.

Theorem 2.1 ([3]). SCP is strongly NP-complete. □
2.2 The Theory of Demand Bound Functions

The hardness proof we present in the next section relies heavily on demand bound functions, and in particular the following, well-known theorem due to Baruah et al.

**Theorem 2.2 ([1, 3]).** A sporadic task system $T$ is feasible on a preemptive uniprocessor if and only if $U(T) \leq 1$ and

$$\forall \ell \geq 0, \quad dbf(T, \ell) \leq \ell, \quad (2)$$

where

$$dbf(T, \ell) \overset{\text{def}}{=} \sum_{(e,d,p) \in T} \left( \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right) \cdot e \quad (3)$$

is the demand bound function of $T$ in interval lengths $\ell$. □

As a notational convenience, let $dbf(\tau, \ell) \overset{\text{def}}{=} dbf(\{\tau\}, \ell)$ for any task $\tau$.

One of the things that Baruah et al. [1] show using this theorem is that the sporadic feasibility problem is in coNP. A witness to the infeasibility of a task system is simply an $\ell \geq 0$ such that the formula in Eq. (2) is false (if there exists a witness, we are guaranteed that there is also one in $\mathbb{N}$ representable with polynomially many bits). A similar argument can be made for asynchronous periodic tasks.

**Theorem 2.3 ([1, 3]).** The feasibility problem for periodic tasks, both synchronous and asynchronous, is in coNP. □

3 The Hardness of Sporadic Feasibility

Here we will show the strong coNP-hardness of the feasibility problem for sporadic tasks on preemptive uniprocessors. This is achieved by means of a pseudo-polynomial transformation (as defined by Garey and Johnson [6]) from SCP to the complement of the feasibility problem.

3.1 Overview of the Transformation

First we describe the intuition behind the transformation. In Figure 2 we have marked along a number line the $x \in \mathbb{N}$ such that $x \equiv a_i \pmod{b_i}$, for four example pairs $(a_i, b_i)$.

We would like to somehow match the structure of these congruence classes with demand bound functions. For each pair $(a_i, b_i)$ we want to create a demand bound function (in interval lengths $\ell$) that is highly regular, but has “hard points” of slightly increased demand at those $\ell$ that in some given way are related to the congruence class of $a_i$ modulo $b_i$. If we have two such functions, their hard points should align at exactly those $\ell$ related to both congruences.
classes. Figure 3 illustrates this basic idea. The goal is to take any instance \((A, k)\) of SCP and create \(|A|\) such functions that, when summed, result in a violation of Eq. (2) at some \(\ell\) if and only if at least \(k\) of them align such hard points at \(\ell\).

\[(2, 4)\]
\[(4, 6)\]
\[(3, 8)\]
\[(0, 3)\]

\(\text{Figure 2. Congruence classes } a_i \text{ modulo } b_i \text{ for the four different pairs } (a_i, b_i) \in A, \text{ where } A = \{(2, 4), (4, 6), (3, 8), (0, 3)\}. \text{ It is clear from the figure that there are several } x \in \mathbb{N} \text{ belonging to two congruence classes simultaneously, but it can be shown that there is no } x \text{ belonging to three. Thus, } (A, 2) \text{ is a yes-instance and } (A, 3) \text{ is a no-instance of SCP.}

\[(2, 4)\]
\[(4, 6)\]

\(\text{Figure 3. Conceptual demand bound functions corresponding to the two pairs } (2, 4) \text{ and } (4, 6) \text{ from } A. \text{ The marked areas are where we want slightly increased demand. Note that these functions do not match exactly those that we will get from the transformation.}\)
3.2 Encoding into Task Systems

We now show how the high-level idea for a transformation that was presented in the last section is encoded with actual task systems. It does not appear possible to capture the structure of a congruence class in a way that achieves our goals using only a single sporadic task. We can, however, create a set of tasks that have the sought structure in its (joint) demand bound function. The details of this follow.

Definition 3.1 (Transformation from SCP to (in-)feasibility). The transformation takes any instance \((A, k)\) of SCP, where \(A = \{(a_1, b_1), \ldots, (a_n, b_n)\}\), and produces a sporadic task system \(T_{(A, k)}\).

For each \((a_i, b_i) \in A\), we create the following set of \(b_i\) constrained-deadline sporadic tasks:

\[
T_{(a_i, b_i)} \overset{\text{def}}{=} \left\{ \tau^y_{(a_i, b_i)} \mid y \in \{1, \ldots, b_i\} \right\},
\]

where

\[
\tau^y_{(a_i, b_i)} \overset{\text{def}}{=} \begin{cases} 
(1, a_i n + k - 1, b_i n), & \text{if } y = a_i + 1, \\
(1, y n, b_i n), & \text{otherwise}.
\end{cases}
\]  

(4)

The multiset

\[
T_{(A, k)} \overset{\text{def}}{=} \bigoplus_{(a_i, b_i) \in A} T_{(a_i, b_i)}
\]  

(5)

is the produced task system.

Each task set \(T_{(a_i, b_i)}\) has a highly regular demand bound function, where the hard points are encoded in the slightly shorter deadline of the task \(\tau^{a_i+1}_{(a_i, b_i)}\). Figure 4 shows the demand bound function of a generated set of tasks \(T_{(a_i, b_i)}\), corresponding to some \((a_i, b_i) \in A\). It can be noted that it has the same general structure as the conceptual functions shown in Figure 3.

Note that the number of tasks in \(T_{(A, k)}\), as well as the values of their parameters, are bounded by some two-variable polynomial in the size and the maximum numerical value found in the corresponding SCP instance \((A, k)\). Also, the transformation can trivially be computed in time bounded by such a polynomial. To establish that the above is a valid pseudo-polynomial transformation, what remains to be shown is that \(T_{(A, k)}\) is a no-instance of the feasibility problem if and only if \((A, k)\) is a yes-instance of SCP. We show this in the next section.
Figure 4. The demand bound functions for the tasks in $T_{(2,4)}$, generated from an SCP instance $(A, k)$ where $|A| = n = 4$ and $k = 3$. The top four functions are the demand bound functions for the individual tasks. The dotted function is for the task $\tau^3_{(2,4)}$, which is defined by the special case in Eq. (4). At the bottom is the sum of their demand bound functions, $dbf(T_{(2,4)}, \ell)$. Note that $dbf(T_{(2,4)}, \ell)$ has fixed size steps of width $n$, except for those corresponding to steps of $dbf(\tau^3_{(2,4)}, \ell)$, which occur only $k - 1$ points away from the preceding steps. These shorter steps make up the “hard points” that we envisioned earlier.
3.3 Correctness of the Transformation

Before showing that the transformation in Definition 3.1 is correct, we need to prove two auxiliary lemmas about the characteristics of the demand bound functions of the generated task systems. The first lemma is about the identity property of the demand bound functions at all points $\ell \in \{0, n, 2n, 3n, \ldots \}$.

**Lemma 3.1 (Identity of demand).** Let $(A, k)$ be any SCP instance, where $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$, and let $x \in \mathbb{N}$ be any natural number. Then,

$$\text{dbf}(T(A, k), xn) = xn.$$  

**Proof.** Consider any pair $(a_i, b_i) \in A$. By construction, all tasks $(e, d, p) \in T(a_i, b_i)$ have $e = 1$ and $p = b_i n$. By putting these values in Eq. (3) we get

$$\text{dbf}(T(a_i, b_i), xn) = \sum_{(e,d,p) \in T(a_i, b_i)} \left( \left\lfloor \frac{xn - d}{b_i n} \right\rfloor + 1 \right).$$

Let $\omega$ be the remainder of $x$ divided by $b_i$,

$$\omega \overset{\text{def}}{=} x - \left\lfloor \frac{x}{b_i} \right\rfloor b_i.$$  

Now, take any of the tasks $(e, d, p) \in T(a_i, b_i)$, and observe that because $d \leq p = b_i n$, we have

$$\left\lfloor \frac{xn - d}{b_i n} \right\rfloor + 1 = \left\lfloor \frac{x}{b_i} \right\rfloor + 1 + \omega - \left\lfloor \frac{d}{b_i n} \right\rfloor + 1$$

$$= \left\lfloor \frac{x}{b_i} \right\rfloor + 1 + \left( \left\lfloor \frac{xn - d}{b_i n} \right\rfloor + 1 \right)$$

$$= \left\lfloor \frac{x}{b_i} \right\rfloor + 1 + \left\lfloor \frac{\omega n - d}{b_i n} \right\rfloor + 1$$

$$= \left\lfloor \frac{x}{b_i} \right\rfloor + 1, \quad \text{if } d \leq \omega n,$$

$$\left\lfloor \frac{x}{b_i} \right\rfloor, \quad \text{otherwise.}$$

How many of the tasks $(e, d, p) \in T(a_i, b_i)$ have $d \leq \omega n$? From Eq. (4) it is clear that the only tasks $(e, d, p) \in T(a_i, b_i)$ with $d \leq \omega n$ are the tasks $\tau^y (a_i, b_i)$ for which $y \in \{1, \ldots, \omega\}$. Hence, $\omega$ out of the $b_i$ tasks in $T(a_i, b_i)$ have $d \leq \omega n$, and

$$\text{dbf}(T(a_i, b_i), xn) = \left\lfloor \frac{x}{b_i} \right\rfloor (b_i - \omega) + (\left\lfloor \frac{x}{b_i} \right\rfloor + 1) \omega$$

$$= \left\lfloor \frac{x}{b_i} \right\rfloor b_i + \omega$$

$$= \left\lfloor \frac{x}{b_i} \right\rfloor b_i + x - \left\lfloor \frac{x}{b_i} \right\rfloor b_i$$

$$= x.$$ 

158
From Eq. (3), (5) and (6) we conclude that
\[
dbf(T_{(A,k)} , x\,n) = \sum_{(a_i,b_i) \in A} \dbf(T_{(a_i,b_i)} , x\,n) = x\,n,
\]
which is the claim of the lemma.

The second auxiliary lemma is about the increased demand of task sets \( T_{(a_i,b_i)} \) for all interval lengths \( \ell \) that are related to the congruence class of \( a_i \) modulo \( b_i \). In particular, for all \( \ell = x\,n + k - 1 \), where \( x \equiv a_i \mod b_i \).

**Lemma 3.2 (Increased demand at congruences).** Let \((A,k)\) be any SCP instance, where \( A = \{(a_1,b_1), \ldots (a_n,b_n)\} \), and let \( x \in \mathbb{N} \) be any natural number. Then,
\[
dbf(T_{(a_i,b_i)} , x\,n + k - 1) = \begin{cases} x + 1, & \text{if } x \equiv a_i \mod b_i, \\ x, & \text{otherwise}, \end{cases}
\]
for all \((a_i,b_i) \in A\).

**Proof.** Let \( \ell^* \overset{\text{def}}{=} x\,n + k - 1 \). Again, by construction we have \( e = 1 \) and \( p = b_i\,n \) for all \((e,d,p) \in T_{(a_i,b_i)}\), and therefore
\[
dbf(T_{(a_i,b_i)} , \ell^*) = \sum_{(e,d,p) \in T_{(a_i,b_i)}} \left( \left\lfloor \frac{\ell^* - d}{b_i\,n} \right\rfloor + 1 \right).
\]

Let \( \omega \overset{\text{def}}{=} x - \left\lfloor \frac{x}{b_i} \right\rfloor b_i \) be defined as before. Take any \((e,d,p) \in T_{(a_i,b_i)}\) and note that \( d - k + 1 \leq p = b_i\,n \). Hence,
\[
\left\lfloor \frac{\ell^* - d}{b_i\,n} \right\rfloor = \left\lfloor \frac{x\,n - (d - k + 1)}{b_i\,n} \right\rfloor
= \begin{cases} \left\lfloor \frac{x}{b_i} \right\rfloor, & \text{if } d - k + 1 \leq \omega\,n, \\ \left\lfloor \frac{x}{b_i} \right\rfloor - 1, & \text{otherwise}. \end{cases}
\]
We rewrite this to obtain
\[
\left\lfloor \frac{\ell^* - d}{b_i\,n} \right\rfloor + 1 = \begin{cases} \left\lfloor \frac{x}{b_i} \right\rfloor + 1, & \text{if } d \leq \omega\,n + k - 1, \\ \left\lfloor \frac{x}{b_i} \right\rfloor, & \text{otherwise}. \end{cases}
\]
From Eq. (4) it is clear that if \( \omega \neq a_i \), then \( d \leq \omega\,n + k - 1 \) holds for all the tasks \( \tau_{(a_i,b_i)}^y \in T_{(a_i,b_i)} \), such that \( y \leq \omega \). However, if \( \omega = a_i \), then \( d \leq \omega\,n + k - 1 \) additionally holds for \( \tau_{(a_i,b_i)}^{a_i+1} \). Hence, if we let \( \alpha \) denote the number of tasks \((e,d,p) \in T_{(a_i,b_i)}\) for which \( d \leq \omega\,n + k - 1 \) holds, then
\[
\alpha = \begin{cases} \omega + 1, & \text{if } \omega = a_i, \\ \omega, & \text{otherwise}. \end{cases}
\]
By the definition of \( \omega \) as the remainder of \( x \) divided by \( b_i \), we have \( \omega = a_i \) if and only if \( x \equiv a_i \pmod{b_i} \). Thus, we can rewrite the above as

\[
\alpha = \begin{cases} 
  x - \lfloor x/b_i \rfloor b_i + 1, & \text{if } x \equiv a_i \pmod{b_i}, \\
  x - \lfloor x/b_i \rfloor b_i, & \text{otherwise}.
\end{cases}
\]

By applying similar steps as in Eq. (6), we conclude that

\[
dbf(T_{(a_i, b_i)}, \ell^*) = \lfloor x/b_i \rfloor (b_i - \alpha) + (\lfloor x/b_i \rfloor + 1) \alpha \\
= \lfloor x/b_i \rfloor b_i + \alpha \\
= \begin{cases} 
  x + 1, & \text{if } x \equiv a_i \pmod{b_i}, \\
  x, & \text{otherwise},
\end{cases}
\]

from which the lemma follows. \( \blacksquare \)

We can now prove the correctness of the transformation.

**Lemma 3.3 (Validity of transformation).** For any SCP instance \((A, k)\), the corresponding task system \(T_{(A, k)}\) is infeasible if and only if \((A, k)\) is a yes-instance.

**Proof.** Let \( A = \{(a_1, b_1), \ldots, (a_n, b_n)\} \). First we note that \( U(T_{(a_i, b_i)}) = 1/n \) for any \((a_i, b_i) \in A\), and consequently that \( U(T_{(A, k)}) = 1 \). By Theorem 2.2 it follows that the feasibility of \( T_{(A, k)} \) is exactly decided by the truth value of the formula in Eq. (2).

We prove the two directions of the lemma separately, beginning with the if-case. Figure 5 serves as an illustration.

- \((A, k)\) is a yes-instance \(\Rightarrow T_{(A, k)}\) is infeasible:
  
  By assumption, there is a subset \( A' \subseteq A \) of size at least \( k \) and an \( x \in \mathbb{N} \), such that \( x \equiv a_i \pmod{b_i} \) for all \((a_i, b_i) \in A'\). Without loss of generality, let that \( A' \) be the largest such subset.

  Let \( \ell^* \defeq xn + k - 1 \). By Lemma 3.2 and the above assumptions, we have

  \[
  \dbf(T_{(a_i, b_i)}, \ell^*) = \begin{cases} 
    x + 1, & \text{if } (a_i, b_i) \in A', \\
    x, & \text{otherwise}.
  \end{cases}
  \]

  Because \( |A'| \geq k \), it follows that

  \[
  \dbf(T_{(A, k)}, \ell^*) = \sum_{(a_i, b_i) \in A} \dbf(T_{(a_i, b_i)}, \ell^*) \\
  \geq x + k > \ell^*,
  \]

  and \( T_{(A, k)} \) is infeasible by Theorem 2.2.
• $(A, k)$ is a no-instance $\Rightarrow T_{(A,k)}$ is feasible:

From Eq. (3) it is clear that $\text{dbf}(T_{(A,k)}, \ell)$ is a right-continuous, non-decreasing step function in $\ell$. Let $\Delta$ be the set of points at which this function changes value, including point 0:

$$\Delta \overset{\text{def}}{=} \{ \ell \mid \text{dbf}(T_{(A,k)}, \ell) \text{ is discontinuous at } \ell \} \cup \{0\}$$

It is easily seen that if there exists some $\ell \geq 0$ such that $\text{dbf}(T_{(A,k)}, \ell) > \ell$, then there must exist some $\ell' \in \Delta$ such that $\text{dbf}(T_{(A,k)}, \ell') > \ell'$. Hence, by showing that $\text{dbf}(T_{(A,k)}, \ell) \leq \ell$ for all $\ell \in \Delta$, we can conclude that there exists no $\ell \geq 0$ such that $\text{dbf}(T_{(A,k)}, \ell) > \ell$. In order to do so we need to find a more concrete characterization of $\Delta$.

Note that from Eq. (4) we know that for all tasks $(e, d, p) \in T_{(A,k)}$ we have

$$p = bn, \text{ for some } b \in \mathbb{N},$$

$$d \in \{yn, yn + k - 1\}, \text{ for some } y \in \mathbb{N}.$$  

By the definition of demand bound functions given in Eq. (3), it follows that the points at which $\text{dbf}(T_{(A,k)}, \ell)$ is discontinuous are of the form $xn$ or $xn + k - 1$ for some $x \in \mathbb{N}$, and therefore

$$\Delta \subseteq \{xn \mid x \in \mathbb{N}\} \cup \{xn + k - 1 \mid x \in \mathbb{N}\}.$$ 

From Lemma 3.1 we directly have

$$\text{dbf}(T_{(A,k)}, \ell) = \ell, \text{ for all } \ell \in \{xn \mid x \in \mathbb{N}\}.$$ 

Consider instead any $\ell^* = xn + k - 1$, where $x \in \mathbb{N}$. By Lemma 3.2, we know that for any $(a_i, b_i) \in A$,

$$\text{dbf}(T_{(a_i, b_i)}, \ell^*) = \begin{cases} 
    x + 1, & \text{if } x \equiv a_i \pmod{b_i}, \\
    x, & \text{otherwise},
\end{cases}$$ 

Now, by assumption, $(A, k)$ is a no-instance of SCP. It follows that there are at most $k - 1$ pairs $(a_i, b_i) \in A$ such that $x \equiv a_i \pmod{b_i}$. Hence,

$$\text{dbf}(T_{(A,k)}, \ell^*) = \sum_{(a_i, b_i) \in A} \text{dbf}(T_{(a_i, b_i)}, \ell^*) \leq xn + k - 1 = \ell^*,$$

for all $\ell^* \in \{xn + k - 1 \mid x \in \mathbb{N}\}$.

In conclusion, we have shown that $\text{dbf}(T_{(A,k)}, \ell) \leq \ell$ for all $\ell \in \Delta$, and consequently for all $\ell \geq 0$. The feasibility of $T_{(A,k)}$ is ensured by Theorem 2.2.
Figure 5. Demand bound functions for task sets generated from an SCP instance $(A, k)$, where $A = \{(2, 4), (4, 6), (3, 8), (0, 3)\}$ and $k = 2$. As we have seen in Figure 2, $(A, 2)$ is a yes-instance of SCP and therefore $T_{(A, 2)}$ should be infeasible. Indeed, for any $x$ belonging to two of $A$'s congruence classes (such as $x = 10$) we have two “early” steps in the corresponding demand bound functions aligning at $\ell^* = xn + 2 - 1$, and $\text{dbf}(T_{(A,2)}, \ell^*) = \ell^* + 1$. If instead $k = 3$, then the early steps would move one unit to the right in the figures, and two such steps aligning at some $\ell^* = xn + 3 - 1$ is no longer enough to witness infeasibility, because $\text{dbf}(T_{(A,3)}, \ell^*) = \ell^*$. 
Our main theorem follows.

**Theorem 3.4** (Intractability). *Deciding if a sporadic task system with constrained deadlines is feasible on a preemptive uniprocessor is coNP-complete in the strong sense.*

*Proof.* There is a pseudo-polynomial transformation from SCP to the complement of the feasibility problem. By Theorems 2.1 and 2.3 we know that SCP is strongly NP-hard and that the feasibility problem is in coNP. 

4 Conclusions

We have showed that there can be no pseudo-polynomial time algorithm for deciding the feasibility of constrained-deadline sporadic task systems on a preemptive uniprocessor, unless \( P = NP \).

This highlights the inherent practical importance of the special case where the utilization of the task system is a priori bounded by some constant \( c < 1 \), for which Baruah et al. [1] have described a pseudo-polynomial time feasibility test. This test (or variations thereof) is widely used in the literature and has proven to be quite tractable, at least for off-line analysis, even for large \( c \) such as 0.9 or 0.95. An important outstanding question is whether this special case also has a polynomial time solution.

References


Uniprocessor Feasibility of Sporadic Tasks Remains \( \text{coNP} \)-complete Under Bounded Utilization

Pontus Ekberg and Wang Yi

Abstract

A central problem in real-time scheduling theory is to decide whether a sporadic task system with constrained deadlines is feasible on a preemptive uniprocessor. It is known that this problem is strongly \( \text{coNP} \)-complete in the general case, but also that there exists a pseudo-polynomial time solution for instances with utilization bounded from above by any constant \( c \), where \( 0 < c < 1 \). For a long time it has been unknown whether the bounded case also has a polynomial-time solution. We show that for any choice of the constant \( c \), such that \( 0 < c < 1 \), the bounded feasibility problem is (weakly) \( \text{coNP} \)-complete, and thus that no polynomial-time solution exists for it, unless \( P = \text{NP} \).

1 Introduction

We let a sporadic task system be defined as a finite multiset \( T \) of tasks, where each task is a triple \( (e, d, p) \in \mathbb{N}_0^3 \), representing its worst-case execution time, relative deadline and minimum inter-arrival separation (or period), respectively.

A sporadic task generates potentially unbounded sequences of jobs. A job is an instance of the task’s workload, characterized by a release time, an execution time and an absolute deadline. A job from task \( (e, d, p) \) has execution time not larger than \( e \) time units and absolute deadline exactly \( d \) time units after its release time. Release times of two consecutive jobs from \( (e, d, p) \) are separated by at least \( p \) time units. A sporadic task system \( T \) generates any interleaving of job sequences that can be generated by each of the tasks \( (e, d, p) \in T \).

For a sequence of jobs to be successfully scheduled, every job must be executed for a total duration equal to its execution time, between its release time and its absolute deadline. In the following we assume a preemptive uniprocessor, meaning that only one job can be executed at a time, but a job can be paused and resumed at a later time at no additional cost.
**Definition 1.1** (Feasibility). A task system $T$ is **feasible** if and only if there is some scheduling algorithm that will successfully schedule all job sequences that can be generated by $T$. □

A task system $T$ is said to have **implicit deadlines** if $d = p$ for all $(e, d, p) \in T$, and said to have **constrained deadlines** if $d \leq p$ for all $(e, d, p) \in T$.

The **utilization** of a task system $T$ is a measure of its asymptotic resource requirements, and is defined as

$$U(T) \overset{\text{def}}{=} \sum_{(e, d, p) \in T} \frac{e}{p}.$$  (1)

Liu and Layland [8] showed that a task system $T$ with implicit deadlines is feasible if and only if $U(T) \leq 1$, but this is not a sufficient condition with constrained deadlines.

Dertouzos [4] has shown that Earliest Deadline First (EDF) is an optimal scheduling algorithm on preemptive uniprocessors, for all sequences of independent jobs. This means that the terms “feasibility” and “EDF-schedulability” can be used interchangeably here.

**Theorem 1.1** ([4]). A task system $T$ is feasible if and only if it is EDF-schedulable. □

A related workload model is that of (strictly) periodic tasks. A periodic task system is a multiset of quadruples $(s, e, d, p) \in \mathbb{N} \times \mathbb{N}^3_+$. The difference to the sporadic task model is that the release times of two consecutive jobs from task $(s, e, d, p)$ must be exactly $p$ time units apart, and the release time of the first job is fixed at time point $s$. A periodic task system is **synchronous** if $s = 0$ for all tasks $(s, e, d, p)$, and **asynchronous** otherwise. It is known [1] that feasibility testing of sporadic tasks is equally hard as that of synchronous periodic tasks, which means that the terms “sporadic” and “synchronous periodic” can be used interchangeably for the results in this paper as well.

**Theorem 1.2** ([1]). A sporadic task system $T$ is feasible if and only if the synchronous periodic task system $\{(0, e, d, p) \mid (e, d, p) \in T\}$ is feasible. □

A classic result [7, 3] is that the feasibility problem for asynchronous periodic task systems with constrained deadlines is strongly coNP-complete\(^1\), even if restricted to instances where the utilization is bounded from above by a constant $c$, such that $0 < c < 1$. However, Baruah et al. [3, 1] found a pseudo-polynomial time algorithm for the special case of synchronous periodic (or sporadic) task systems with such a priori bounded utilization.

Except for the existence of this pseudo-polynomial time algorithm, the complexity of the sporadic feasibility problem remained open for many years.

\(^1\)Recall that the problems that are NP- or coNP-complete in the strong sense do not permit even pseudo-polynomial time solutions, unless $P = NP$, while those that are NP- or coNP-complete in the weak (or ordinary) sense might have such solutions.
It was listed as one of the “most important open algorithmic problems in real-time scheduling” by Baruah and Pruhs [2]. A few years ago, Eisenbrand and Rothvoß [5] showed that the general case (with unbounded utilization) is weakly coNP-complete. This was recently strengthened by Ekberg and Yi [6], who showed that the general case is strongly coNP-complete.

**Theorem 1.3 ([6]).** Deciding whether a sporadic task system with constrained deadlines is feasible on a preemptive uniprocessor is coNP-complete in the strong sense.

For the case with bounded utilization, it was still unknown if a polynomial-time solution existed, and Eisenbrand and Rothvoß [5] conjectured that it did. In this paper, however, we show that for any choice of the constant $c$, such that $0 < c < 1$, the feasibility problem for sporadic task systems restricted to instances with utilization at most $c$ is weakly coNP-complete, and thus that no polynomial-time solutions exist for it, unless $P = NP$. As it is known that this problem has a pseudo-polynomial time solution, this is a complete classification. A summary of the complexity of the periodic feasibility problem is shown in Figure 1.

<table>
<thead>
<tr>
<th>Asynchronous periodic tasks</th>
<th>Utilization bounded by some constant $c$, such that $0 &lt; c &lt; 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>coNP-complete in the strong sense [3]</td>
<td>coNP-complete in the strong sense [3]</td>
</tr>
<tr>
<td>Synchronous periodic tasks (or sporadic)</td>
<td>coNP-complete in the strong sense [6]</td>
</tr>
</tbody>
</table>

**Figure 1.** Complexity classification of the feasibility problem of periodic tasks with constrained deadlines on preemptive uniprocessors.

Let FEASIBILITY denote the general case of the decision problem of whether a given sporadic task system with constrained deadlines is feasible on a preemptive uniprocessor. Let $c$-FEASIBILITY denote the same problem restricted to instances with utilization at most $c$. Our hardness proof consists of a polynomial transformation (a polynomial-time many-one reduction) from FEASIBILITY to $c$-FEASIBILITY for any constant $c$, such that $0 < c < 1$. The transformation produces some parameter values that grow exponentially in the size of the FEASIBILITY instance. Therefore, $c$-FEASIBILITY is only shown to be
weakly coNP-hard (as expected, since a pseudo-polynomial time solution is known) despite the strong coNP-hardness of FEASIBILITY.

2 Preliminaries

In the hardness proof in the next section we will make use of the following theorem due to Baruah et al. [1]

Theorem 2.1 ([1]). A sporadic task system $T$ with constrained deadlines is feasible on a preemptive uniprocessor if and only if $U(T) \leq 1$ and

$$\forall \ell \in \{0, 1, \ldots, B\}, \quad dbf(T, \ell) \leq \ell,$$

where $B = \mathcal{P}(T) + \max\{d \mid (e, d, p) \in T\}$, and where

$$dbf(T, \ell) \triangleq \sum_{(e, d, p) \in T} \left( \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right) e$$

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

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

is $T$’s hyper-period. □

For our purposes it will be helpful to reformulate Eq. (2) in the above theorem in terms of slack and slightly tighten the upper bound $B$ used for the values of $\ell$. First we prove a simple lemma that will also be useful later on.

Lemma 2.2. If $T$ is a sporadic task system with constrained deadlines and $k \in \mathbb{N}$, then

$$dbf(T, k\mathcal{P}(T) + \ell) = k\mathcal{P}(T)U(T) + dbf(T, \ell).$$

Proof. From the fact that each of the periods divides the hyper-period, we get

$$dbf(T, k\mathcal{P}(T) + \ell) = \sum_{(e, d, p) \in T} \left( k\mathcal{P}(T) + \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right) e$$

$$= \sum_{(e, d, p) \in T} \left( k\mathcal{P}(T) + \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right) e$$

$$= \sum_{(e, d, p) \in T} \left( ek\mathcal{P}(T) + \left( \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right) e \right)$$

$$= k\mathcal{P}(T)U(T) + dbf(T, \ell).$$

2In [1], the considered tasks could have relative deadlines larger than their periods, which is why the larger value $B$ is used there.

170
As we reformulate Theorem 2.1, we use this lemma to tighten the bound on the values of \( \ell \) that we need to consider.

**Corollary 2.3.** A sporadic task system \( T \) with constrained deadlines is feasible on a preemptive uniprocessor if and only if

\[
\forall \ell \in \{0, 1, \ldots, P(T) - 1\}, \quad \text{slack}(T, \ell) \geq 0,
\]

where

\[
\text{slack}(T, \ell) \overset{\text{def}}{=} \ell - \text{dbf}(T, \ell).
\]

**Proof.** To see that the smaller range of values for \( \ell \) considered in Eq. (5) is sufficient, note that from Lemma 2.2 and \( U(T) \leq 1 \) we have, for any \( k \in \mathbb{N} \),

\[
\text{dbf}(T, kP(T) + \ell) = kP(T)U(T) + \text{dbf}(T, \ell) \leq kP(T) + \text{dbf}(T, \ell).
\]

Therefore,

\[
\text{slack}(T, kP(T) + \ell) = kP(T) + \ell - \text{dbf}(T, kP(T) + \ell) \\
\geq kP(T) + \ell - kP(T) - \text{dbf}(T, \ell) \\
= \text{slack}(T, \ell)
\]

and so if \( \text{slack}(T, \ell) < 0 \) holds for any \( \ell \in \mathbb{N} \), then it must be the case that \( \text{slack}(T, \ell \mod P(T)) < 0 \) also holds. ■

# 3 Reducing **Feasibility** to **c-Feasibility**

In this section we describe a polynomial transformation from **Feasibility** to **c-Feasibility**, for any given constant \( c \), such that \( 0 < c < 1 \). We start by providing a high-level picture of the transformation in Section 3.1, and follow with the details and proofs in Section 3.2.

## 3.1 An overview

For the transformation we take an arbitrary sporadic task system \( T_1 \) with constrained deadlines and compute a new task system \( T_c \), also with constrained deadlines, such that

- \( U(T_c) \leq c \) and
- \( T_c \) is feasible if and only if \( T_1 \) is feasible.

Note that if \( U(T_1) > 1 \), then \( T_1 \) is always infeasible and the transformation is trivially computable by producing any infeasible \( T_c \), for example \( T_c = \{(1, 1, \ceil{2/c}), (1, 1, \ceil{2/c})\} \). In the remainder of this paper we assume \( U(T_1) \leq 1 \), without loss of generality.
The construction of \( T_c \) follows these four high-level steps.

**Step 1:** Add a filler task to \( T_1 \), resulting in task system \( T_{\text{fill}} \), without affecting feasibility. The purpose is to ensure that \( \text{slack}(T_{\text{fill}}, \ell) \) exactly repeats every hyper-period.

**Step 2:** Scale the deadlines and periods of \( T_{\text{fill}} \) uniformly with a given constant factor \( \sigma \), producing task system \( T_{\sigma} \). The purpose is to lower utilization while keeping a simple relation between \( \text{slack}(T_{\text{fill}}, \ell) \) and \( \text{slack}(T_{\sigma}, \ell) \).

**Step 3:** Construct a “boosting” task system \( T_b \), which is designed to fill up the extra slack created by the scaling in step 2 at certain sparsely selected points \( \ell \). The purpose is to effectively negate the scaling done in step 2 without increasing the utilization too much. The extra slack is filled up at points that are offset by different multiples of \( T_{\sigma} \)'s hyper-period. This is possible thanks to the repetitive nature of \( \text{slack}(T_{\sigma}, \ell) \) that we ensured in step 1.

**Step 4:** Take \( T_c \) as the union of \( T_{\sigma} \) and \( T_b \). Even though \( U(T_c) \leq c \), we will have that there exists some \( \ell \in \mathbb{N} \), such that \( \text{slack}(T_1, \ell) < 0 \) if and only if there exists some \( \ell' \in \mathbb{N} \), such that \( \text{slack}(T_c, \ell') < 0 \).

### 3.2 The transformation

Here we describe the transformation in detail and prove a number of lemmas along the way.

**► Step 1:** *Ensuring the repetitiveness of the slack function*

In the first step we simply add a filler task \( \tau_{\text{fill}} \) to \( T_1 \), if needed, to create a new task system \( T_{\text{fill}} \) with a utilization of 1. Figure 2 illustrates this step. Let

\[
T_{\text{fill}} \overset{\text{def}}{=} \begin{cases} 
T_1 \cup \{\tau_{\text{fill}}\}, & \text{if } U(T_1) < 1, \\
T_1, & \text{otherwise,}
\end{cases}
\]  

(7)

where \( \tau_{\text{fill}} \overset{\text{def}}{=} (\mathcal{P}(T_1) - \text{dbf}(T_1, \mathcal{P}(T_1)), \mathcal{P}(T_1), \mathcal{P}(T_1)) \) and where \( \cup \) is the multiset union operator.
Figure 2. The demand bound functions of a small example task system $T_1$ and the corresponding $T_{\text{fill}}$. Note that the difference between the diagonal and $\text{dbf}(T_{\text{fill}}, \ell)$, which is $\text{slack}(T_{\text{fill}}, \ell)$, repeats every hyper-period.

Lemma 3.1. $U(T_{\text{fill}}) = 1$.

Proof. If $U(T_1) = 1$, the lemma follows directly. Assume instead $U(T_1) < 1$ and note that

$$U(\{\tau_{\text{fill}}\}) = \frac{\mathcal{P}(T_1) - \text{dbf}(T_1, \mathcal{P}(T_1))}{\mathcal{P}(T_1)}$$

$$= 1 - \frac{\sum_{(e,d,p) \in T_1} \left( \left\lfloor \frac{\mathcal{P}(T_1) - d}{p} \right\rfloor + 1 \right) e}{\mathcal{P}(T_1)}$$

$$= 1 - \frac{\sum_{(e,d,p) \in T_1} \left( e\mathcal{P}(T_1) \frac{p}{p} + \left( \left\lfloor \frac{-d}{p} \right\rfloor + 1 \right) e \right)}{\mathcal{P}(T_1)}$$

$$\overset{(*)}{=} 1 - U(T_1),$$

where for $(*)$ we used the fact that $\left\lfloor \frac{-d}{p} \right\rfloor = -1$. Clearly, $U(T_{\text{fill}}) = U(T_1) + U(\{\tau_{\text{fill}}\}) = 1.$
Now, the property of \( T_{\text{fill}} \) that will be useful for our transformation is captured by the next lemma.

**Lemma 3.2.** For any \( \ell \in \{0, 1, \ldots, P(T_1) - 1\} \) and \( k \in \mathbb{N} \),

\[
\text{slack}(T_{\text{fill}}, kP(T_1) + \ell) = \text{slack}(T_1, \ell).
\]

**Proof.** Note that \( P(T_{\text{fill}}) = P(T_1) \), and therefore it follows from Lemmas 2.2 and 3.1 that

\[
\begin{align*}
\text{slack}(T_{\text{fill}}, kP(T_1) + \ell) & = kP(T_1) + \ell - \text{dbf}(T_{\text{fill}}, kP(T_1) + \ell) \\
& = kP(T_1) + \ell - (kP(T_1)U(T_{\text{fill}}) + \text{dbf}(T_{\text{fill}}, \ell)) \\
& = \ell - \text{dbf}(T_{\text{fill}}, \ell) \\
& \overset{(*)}{=} \ell - \text{dbf}(T_1, \ell) \\
& = \text{slack}(T_1, \ell),
\end{align*}
\]

where for (\( * \)) we have \( \text{dbf}(T_{\text{fill}}, \ell) = \text{dbf}(T_1, \ell) \) because \( T_{\text{fill}} \) and \( T_1 \) at most differ on one task \( \tau_{\text{fill}} = (e,d,p) \), for which \( d = P(T_1) > \ell \) and therefore \( \text{dbf}(\{\tau_{\text{fill}}\}, \ell) = 0 \).  

---

**Step 2: Scaling down the utilization**

In the second step we apply a scaling factor to the relative deadlines and periods of all the tasks in \( T_{\text{fill}} \). Figure 3 serves as an illustration. Let

\[
\sigma \overset{\text{def}}{=} \left\lceil \frac{2}{c} \right\rceil \tag{8}
\]

be the constant scaling factor, where \( c \) is the constant defining the set of problems \( c\text{-FEASIBILITY} \). Then let

\[
T_\sigma \overset{\text{def}}{=} \{(e, \sigma d, \sigma p) \mid (e, d, p) \in T_{\text{fill}}\} \tag{9}
\]

be the scaled task system. The utilization of \( T_\sigma \) is now bounded by half of \( c \).

**Lemma 3.3.** \( U(T_\sigma) \leq c/2 \).

**Proof.** Directly from Lemma 3.1, Eq. (8) and (9).
Lemma 3.4. For any $\ell \in \mathbb{N}$,

$$\text{slack}(T_\sigma, \sigma \ell) = \text{slack}(T_{\text{fill}}, \ell) + (\sigma - 1)\ell.$$  

Proof. From the definition of $T_\sigma$ we get

$$\text{slack}(T_\sigma, \sigma \ell) = \sigma \ell - \text{dbf}(T_\sigma, \sigma \ell)$$

$$= \sigma \ell - \sum_{(e,d,p) \in T_\sigma} \left( \left\lfloor \frac{\sigma \ell - d}{p} \right\rfloor + 1 \right) e$$

$$= \sigma \ell - \sum_{(e,d,p) \in T_{\text{fill}}} \left( \left\lfloor \frac{\sigma \ell - \sigma d}{\sigma p} \right\rfloor + 1 \right) e$$

$$= \sigma \ell - \sum_{(e,d,p) \in T_{\text{fill}}} \left( \left\lfloor \frac{\ell - d}{p} \right\rfloor + 1 \right) e$$

$$= \sigma \ell - \text{dbf}(T_{\text{fill}}, \ell)$$

$$= \text{slack}(T_{\text{fill}}, \ell) + (\sigma - 1)\ell.$$

$\blacksquare$
Step 3: Generating a boosting task system

The task system $T_{\sigma}$ created in the last step clearly does not preserve feasibility with regards to $T_{\text{fill}}$ because of the extra slack introduced by the scaling of parameters. In this step we will craft a task system $T_b$, designed to effectively negate this extra slack. The main challenge is in doing so while using only polynomially many tasks and keeping the utilization low enough. This step is more involved than the previous steps and also a bit harder to visualize, but Figure 4 illustrates some of the key concepts. The boosting task system will contain $\beta$ different tasks that are referred to using their indices,

$$T_b \overset{\text{def}}{=} \{ \tau_0, \ldots, \tau_{\beta-1} \},$$

where

$$\beta \overset{\text{def}}{=} \left\lceil \log_2(\mathcal{P}(T_1)) \right\rceil.$$  

For each boosting task $\tau_i$, let $\tau_i \overset{\text{def}}{=} (e_i, d_i, p_i)$, where

$$d_i \overset{\text{def}}{=} (\sigma \mathcal{P}(T_1) + 2)^i \sigma,$$

$$p_i \overset{\text{def}}{=} (\sigma \mathcal{P}(T_1) + 2)^{i+1} \sigma,$$

$$e_i \overset{\text{def}}{=} \left( \frac{\sigma - 1}{\sigma} - \sum_{j=0}^{i-1} \frac{e_j}{p_j} \right) d_i.$$  

The first thing we need to show is that the boosting task system, as defined above, consists of valid constrained-deadline sporadic tasks. The next lemma establishes this and some slightly stronger constraints on the parameters.

Lemma 3.5. For all $\tau_i \in T_b$, we have

$$(e_i, d_i, p_i) \in \mathbb{N}_+^3 \text{ and } e_i < d_i < p_i.$$  

Proof. It follows directly from Eq. (12) and (13) that for any $\tau_i \in T_b$, we have $d_i, p_i \in \mathbb{N}_+$ and $d_i < p_i$. What remains to be shown is that

$$e_i \in \mathbb{N}_+ \text{ and } e_i < d_i.$$  

We use strong induction over $i$ to prove this statement for all $\tau_i \in T_b$. As there is nothing to show for $i \geq \beta$, we assume $i < \beta$ in the following.

For the base case, note that

$$e_0 = \left( \frac{\sigma - 1}{\sigma} \right) d_0 = \left( \frac{\sigma - 1}{\sigma} \right) \sigma = \sigma - 1.$$  

By Eq. (8) we have $\sigma \geq 3$, and therefore $\sigma - 1 \in \mathbb{N}_+$ and $e_0 = \sigma - 1 < \sigma = d_0$.  

176
We split the induction step into three parts, showing that $e_i \in \mathbb{Z}$, that $e_i > 0$ and that $e_i < d_i$, respectively.

**Part 1** ($e_i \in \mathbb{Z}$): First, we rewrite $e_i$ in a form without fractions.

$$e_i = \left( \frac{\sigma - 1}{\sigma} - \sum_{j=0}^{i-1} \frac{e_j}{p_j} \right) d_i$$

$$= (\sigma - 1)(\sigma \mathcal{P}(T_1) + 2)^i - \sum_{j=0}^{i-1} e_j (\sigma \mathcal{P}(T_1) + 2)^{i-j}$$

Now, from the induction hypothesis we have $e_j \in \mathbb{N}_+$ for $j < i$, and therefore the rewritten form of $e_i$ consists of only additions, subtractions and (repeated) multiplications of integers. As $\mathbb{Z}$ are closed under these operations, it follows that $e_i \in \mathbb{Z}$.

**Part 2** ($e_i > 0$): We have

$$e_i = \left( \frac{\sigma - 1}{\sigma} - \sum_{j=0}^{i-1} \frac{e_j}{p_j} \right) d_i$$

$$(*) \quad \geq \left( \frac{\sigma - 1}{\sigma} - \sum_{j=0}^{i-1} \frac{d_j}{p_j} \right) d_i$$

$$= \left( \frac{\sigma - 1}{\sigma} - \sum_{j=0}^{i-1} \frac{1}{(\sigma \mathcal{P}(T_1) + 2)} \right) d_i$$

$$= \left( \frac{\sigma - 1}{\sigma} - \frac{i}{(\sigma \mathcal{P}(T_1) + 2)} \right) d_i$$

$$(**) \quad > \left( \frac{\sigma - 1}{\sigma} - \frac{\mathcal{P}(T_1)}{(\sigma \mathcal{P}(T_1) + 2)} \right) d_i$$

$$> \left( \frac{\sigma - 1}{\sigma} - \frac{1}{\sigma} \right) d_i$$

$$\geq \left( \frac{1}{\sigma} \right) d_i$$

$$> 0,$$

where for $(*)$ we used $e_j < d_j$ from the induction hypothesis and for $(**)$ we used $i < \beta = \lceil \log_2(\mathcal{P}(T_1)) \rceil < \mathcal{P}(T_1)$.
Part 3 ($e_i < d_i$): Lastly, note that

$$e_i = \left( \frac{\sigma - 1}{\sigma} - \frac{\sum_{j=0}^{i-1} e_j}{p_j} \right) d_i$$

$$< \left( 1 - \frac{\sum_{j=0}^{i-1} e_j}{p_j} \right) d_i$$

$$\leq d_i,$$  

where for (*) we used $e_j \in \mathbb{N}_+$ for $j < i$ from the induction hypothesis. Taken together, the three parts imply $e_i \in \mathbb{N}_+$ and $e_i < d_i$, which concludes the proof. 

Having shown that $T_b$ is a valid sporadic task system, we can show that it has the properties we seek. We start by showing that the utilization of $T_b$ is low, namely bounded by half of $c$.

**Lemma 3.6.** $U(T_b) < c/2$.

**Proof.** Consider any $\tau_i \in T_b$ and note that

$$U(\{\tau_i\}) = \frac{e_i}{p_i}$$

$$< \frac{d_i}{p_i}$$

$$< \frac{1}{(\sigma P(T_1))}.$$  

Now, because $P(T_1) \in \mathbb{N}_+$, we have $\lfloor \log_2(P(T_1)) \rfloor < P(T_1)$ and therefore

$$U(T_b) < \frac{\beta}{(\sigma P(T_1))}$$

$$< \frac{1}{\sigma}$$

$$\leq \frac{c}{2}.$$  

Now we study the demand of $T_b$ to see that it provides the right amount of it to selectively fill up the extra slack that was created for $T_\sigma$ by the scaling in step 2. With the next lemma we ensure that the demand of $T_b$ never fills up more than this extra slack.

**Lemma 3.7.** For all $\ell \in \mathbb{N}$,

$$\text{dbf}(T_b, \ell) \leq \frac{\sigma - 1}{\sigma} \ell.$$  

**Proof.** First, let $T_b^m$ denote the subset consisting of the smallest $\min(m, \beta)$ tasks in $T_b$,

$$T_b^m \overset{\text{def}}{=} \{ \tau_i \in T_b \mid i < m \}.$$
Then we use induction over $m$ to prove that
\[
\forall \ell \in \mathbb{N}, \ dbf(T^m_b, \ell) \leq \frac{\sigma - 1}{\sigma} \ell,
\]
for all $m \in \mathbb{N}_+$. Because $dbf(T^m_b, \ell) = dbf(T_b, \ell)$ for $m \geq \beta$, the lemma will follow.

The base case holds as we have $T^1_b = \{\tau_0\}$ and
\[
dbf(\tau_0, \ell) = \left(\left\lceil \frac{\ell - d_0}{p_0} \right\rceil + 1 \right) e_0
\leq \left(\left\lceil \frac{\ell - d_0}{d_0} \right\rceil + 1 \right) e_0
= \left\lfloor \frac{\ell - d_0}{d_0} \right\rfloor e_0
\leq \frac{e_0}{d_0} \ell
= \frac{\sigma - 1}{\sigma} \ell.
\]

For the induction step we need to show for all $m \in \mathbb{N}_+$ that
\[
\forall \ell \in \mathbb{N}, \ dbf(T^m_b, \ell) \leq \frac{\sigma - 1}{\sigma} \ell \quad \Longrightarrow \\
\forall \ell \in \mathbb{N}, \ dbf(T^{m+1}_b, \ell) \leq \frac{\sigma - 1}{\sigma} \ell.
\]

The induction step trivially holds for $m \geq \beta$. Assume instead $m < \beta$ and note that $T^{m+1}_b \setminus T^m_b = \{\tau_m\}$. Take any $\ell \in \mathbb{N}$ and write $\ell = kd_m + \ell'$ for some $k \in \mathbb{N}$ and $\ell' < d_m$. Then,
\[
dbf(\tau_m, \ell) = dbf(\tau_m, kd_m + \ell')
= \left(\left\lceil \frac{kd_m + \ell' - d_m}{p_m} \right\rceil + 1 \right) e_m
\leq \left(\left\lceil \frac{kd_m}{p_m} \right\rceil + 1 \right) e_m
\leq ke_m
= k \left(\frac{\sigma - 1}{\sigma} - \sum_{j=0}^{m-1} \frac{e_j}{p_j} \right) d_m
= kd_m \frac{\sigma - 1}{\sigma} - kd_m U(T^m_b),
\]
where for (*) we used $\ell' < d_m$ and for (**) we used $d_m < p_m$. Now, from Eq. (12) and (13) it is clear that $d_m = p_{m-1} = \mathcal{P}(T_b^m)$. Using Lemma 2.2 and the induction hypothesis we then get

$$dbf(T_b^m, \ell) = \mathcal{P}(T_b^m) + dbf(T_b^m, \ell')$$

$$\leq kd_m U(T_b^m) + \frac{\sigma - 1}{\sigma} \ell'.$$

Putting the above together, we get

$$dbf(T_b^{m+1}, \ell) = dbf(T_b^m, \ell) + dbf(\{\tau_m\}, \ell)$$

$$\leq \frac{\sigma - 1}{\sigma} (kd_m + \ell')$$

$$= \frac{\sigma - 1}{\sigma} \ell.$$

To conclude, we have shown that for all $m \in \mathbb{N}_+$,

$$\forall \ell \in \mathbb{N}, dbf(T_b^m, \ell) \leq \frac{\sigma - 1}{\sigma} \ell.$$

In particular, it holds for $m = \beta$ and the lemma follows.

We now consider the points at which $T_b$ provides exactly the right amount of demand to negate the scaling done for $T_\sigma$ in step 2. We call these the boost points of $T_b$ and denote them $B(T_b)$.

$$B(T_b) \overset{\text{def}}{=} \left\{ \ell \in \mathbb{N} \mid dbf(T_b, \ell) = \frac{\sigma - 1}{\sigma} \ell \right\}$$ (15)

The following lemma lets us identify boost points.

**Lemma 3.8.** If $\tilde{T} \subseteq T_b$, then

$$\sum_{\tau_i \in \tilde{T}} d_i \in B(T_b).$$

**Proof.** Consider the relation $(2^{T_b}, \subseteq)$, i.e., the power set of $T_b$ ordered by strict inclusion, and note that it is well-founded. We prove the lemma using well-founded induction on this relation. To do so we must show that the following implication holds for any $\tilde{T}$, such that $\tilde{T} \subseteq T_b$.

$$\left( \forall \tilde{T}, \text{ s.t. } \tilde{T} \subseteq \tilde{T}, \sum_{\tau_i \in \tilde{T}} d_i \in B(T_b) \right) \implies \sum_{\tau_i \in \tilde{T}} d_i \in B(T_b)$$

We split the proof into two cases.
Case 1 \((\bar{T} = \emptyset)\): Clearly, \(\sum_{\tau_i \in \emptyset} d_i = 0\) and from Eq. (15) we have \(0 \in B(T_b)\).

Case 2 \((\bar{T} \neq \emptyset)\): Now, \(\bar{T}\) must contain a “largest” task \(\tau_\alpha\), where

\[
\alpha \overset{\text{def}}{=} \max \{i \mid \tau_i \in \bar{T}\}
\]

is the largest index of the tasks in \(\bar{T}\). Let

\[
\hat{T} \overset{\text{def}}{=} \bar{T} \setminus \{\tau_\alpha\},
\]

\[
\hat{\ell} \overset{\text{def}}{=} \sum_{\tau_i \in \hat{T}} d_i,
\]

\[
\hat{\ell} \overset{\text{def}}{=} d_\alpha + \hat{\ell}.
\]

Since \(\bar{T} \subset \hat{T}\), we know that \(\hat{\ell} \in B(T_b)\) holds from the induction hypothesis. To conclude the proof we must show that \(\hat{\ell} \in B(T_b)\) holds as well.

Consider now the sets of boosting tasks with indices smaller than \(\alpha\) and at least as large as \(\alpha\), respectively.

\[
T_\alpha^\downarrow \overset{\text{def}}{=} \{\tau_i \in T_b \mid i < \alpha\}
\]

\[
T_\alpha^\uparrow \overset{\text{def}}{=} \{\tau_i \in T_b \mid i \geq \alpha\}
\]

Because \(\bar{T} \subseteq T_\alpha^\downarrow\) we have

\[
\hat{\ell} \leq \sum_{\tau_i \in T_\alpha^\downarrow} d_i
\]

\[
= (\sigma P(T_1) + 2)^0 \sigma + \cdots + (\sigma P(T_1) + 2)^{\alpha - 1} \sigma
\]

\[
\overset{(*)}{=} \sigma \left( \frac{(\sigma P(T_1) + 2)^\alpha - 1}{\sigma P(T_1) + 2 - 1} \right)
\]

\[
< (\sigma P(T_1) + 2)^\alpha \sigma
\]

\[
= d_\alpha,
\]

where for \((*)\) we used the closed form for the sum of a finite geometric series. It follows that for all \(\tau_i \in T_\alpha^\uparrow\) we have \(d_i \geq d_\alpha > \hat{\ell}\) and therefore

\[
dbf(T_\alpha^\uparrow, \hat{\ell}) = 0.
\]

Using this and the induction hypothesis we get

\[
dbf(T_\alpha^\downarrow, \hat{\ell}) = dbf(T_b, \hat{\ell}) - dbf(T_\alpha^\uparrow, \hat{\ell})
\]

\[
= dbf(T_b, \hat{\ell})
\]

\[
= \frac{\sigma - 1}{\sigma} \hat{\ell}.
\]
Now, from Eq. (12) and (13) we know that $p_i$ divides $d_\alpha$ for all $\tau_i \in T_{\alpha \downarrow}$. Therefore,

$$
dbf(T_{\alpha \downarrow}, \dot{\ell}) = dbf(T_{\alpha \downarrow}, d_\alpha + \ddot{\ell}) = \sum_{\tau_i \in T_{\alpha \downarrow}} \left( \left\lfloor \frac{d_\alpha + \ddot{\ell} - d_i}{p_i} \right\rfloor + 1 \right) e_i = \sum_{\tau_i \in T_{\alpha \downarrow}} \left( \frac{d_\alpha}{p_i} + \left\lfloor \frac{\ddot{\ell} - d_i}{p_i} \right\rfloor + 1 \right) e_i = \sum_{\tau_i \in T_{\alpha \downarrow}} \frac{e_i}{p_i} d_\alpha + dbf(T_{\alpha \downarrow}, \dot{\ell}) = \sum_{\tau_i \in T_{\alpha \downarrow}} \frac{e_i}{p_i} d_\alpha + \frac{\sigma - 1}{\sigma} \dot{\ell}
$$

Finally,

$$
dbf(T_b, \dot{\ell}) = dbf(T_{\alpha \uparrow}, \dot{\ell}) + dbf(T_{\alpha \downarrow}, \dot{\ell}) \geq dbf(\{\tau_\alpha\}, \dot{\ell}) + dbf(T_{\alpha \downarrow}, \dot{\ell}) \geq dbf(\{\tau_\alpha\}, d_\alpha) + dbf(T_{\alpha \downarrow}, \dot{\ell}) = e_\alpha + dbf(T_{\alpha \downarrow}, \dot{\ell}) = \frac{\sigma - 1}{\sigma} d_\alpha - \sum_{j=0}^{\sigma-1} \frac{e_j}{p_j} d_\alpha + dbf(T_{\alpha \downarrow}, \dot{\ell}) = \frac{\sigma - 1}{\sigma} d_\alpha - \sum_{\tau_i \in T_{\alpha \downarrow}} \frac{e_i}{p_i} d_\alpha + dbf(T_{\alpha \downarrow}, \dot{\ell}) = \frac{\sigma - 1}{\sigma} (d_\alpha + \ddot{\ell}) = \frac{\sigma - 1}{\sigma} \dot{\ell}
$$

The inequalities above must be strict equalities based on Lemma 3.7, and we conclude that $\dot{\ell} \in B(T_b)$.

Now that we have a lemma for identifying boost points, we can show that $B(T_b)$ cover enough relevant points in $\mathbb{N}$.

**Lemma 3.9.** For all $\ell \in \{0, 1, \ldots, \mathcal{P}(T_1) - 1\}$, there exists some $k \in \mathbb{N}$, such that

$$
\sigma(k \mathcal{P}(T_1) + \ell) \in B(T_b).
$$
Proof. Take any $\ell \in \{0, 1, \ldots, \mathcal{P}(T_1) - 1\}$ and note that $\ell$ can be represented in binary using $\beta = \lceil \log_2(\mathcal{P}(T_1)) \rceil$ bits. Let

$$b_{\beta-1} \cdots b_1 b_0$$

be this binary representation of $\ell$, where $b_0$ is the least significant bit. Consider the set of boosting tasks with indices matching the indices of the positive bits in the binary representation of $\ell$

$$\hat{T} \overset{\text{def}}{=} \{ \tau_i \in T_b \mid b_i = 1 \}.$$ 

For any $\tau_i \in \hat{T}$ we can rewrite

$$d_i = (\sigma \mathcal{P}(T_1) + 2)^i \sigma$$

$$= (\sum_{m=0}^{i} \binom{i}{m} \sigma^m \mathcal{P}(T_1)^m 2^{i-m}) \sigma$$

$$= (\sum_{m=1}^{i} \binom{i}{m} \sigma^m \mathcal{P}(T_1)^m 2^{i-m} + 2^i) \sigma$$

$$= (k_i \mathcal{P}(T_1) + 2^i) \sigma,$$

where

$$k_i = \sum_{m=1}^{i} \binom{i}{m} \sigma^m \mathcal{P}(T_1)^{m-1} 2^{i-m}$$

and where for (*) we used the binomial theorem. Let

$$k \overset{\text{def}}{=} \sum_{\tau_i \in \hat{T}} k_i$$

and note that $k \in \mathbb{N}$. We then have

$$\sum_{\tau_i \in \hat{T}} d_i = \sum_{\tau_i \in \hat{T}} (k_i \mathcal{P}(T_1) + 2^i) \sigma$$

$$= \sigma \left( \sum_{\tau_i \in \hat{T}} k_i \mathcal{P}(T_1) + \sum_{\tau_i \in \hat{T}} 2^i \right)$$

$$= \sigma \left( k \mathcal{P}(T_1) + \sum \{ 2^i \mid b_i = 1 \} \right)$$

$$= \sigma (k \mathcal{P}(T_1) + \ell).$$

To conclude, we note that because $\hat{T} \subseteq T_b$, Lemma 3.8 gives us

$$\sum_{\tau_i \in \hat{T}} d_i = \sigma (k \mathcal{P}(T_1) + \ell) \in B(T_b)$$

and the lemma follows.
Figure 4. A plot of \( \frac{\text{dbf}(T_b, \ell)}{\ell} \) for a small example \( T_b \) with \( \beta = 3 \). Note the logarithmic scale on the horizontal axis. The boost points are those points at which this function touches the line \( \frac{(\sigma - 1)}{\sigma} \). Every task \( \tau_i \in T_b \) adds a new “peak” to the plot, touching the line \( \frac{(\sigma - 1)}{\sigma} \) at \( \ell = d_i \). Immediately after, all previous peaks for tasks \( \tau_j \), with \( j < i \), will repeat, as can be seen in the zoomed in portions of the plot. The number of boost points therefore grows exponentially in \( \beta \). The labels show the value of \( \ell \) at the peaks. Note also how the function converges towards \( U(T_b) \) to the right.

▶ Step 4: Assembling the finished task system

In the last step we put everything together. Let

\[
T_c \overset{\text{def}}{=} T_\sigma \cup T_b,
\]

and note that \( T_c \) is an instance of \( c\text{-FEASIBILITY} \).

**Lemma 3.10.** \( U(T_c) < c \).

**Proof.** Directly from Lemmas 3.3 and 3.6. 

We can now show that the transformation correctly preserves the feasibility of the original task system. We start by showing that \( T_c \) is feasible if \( T_1 \) is.
Lemma 3.11. If $T_1$ is feasible, then $T_c$ is also feasible.

Proof. We prove the lemma by contradiction. Assume for this purpose the negation of the lemma’s claim—that $T_1$ is feasible, but $T_c$ is not. By Corollary 2.3, there must then exist some $\ell \in \mathbb{N}$ such that $\text{slack}(T_c, \ell) < 0$. Since all $(e, d, p) \in T_c$ have $d$ and $p$ that are multiples of $\sigma$, it is evident that there must also exist some $\ell' \in \mathbb{N}$ such that

$$\text{slack}(T_c, \sigma \ell') < 0.$$  

Note that

$$\text{slack}(T_c, \sigma \ell') = \sigma \ell' - \text{dbf}(T_c, \sigma \ell') = \sigma \ell' - \text{dbf}(T_\sigma, \sigma \ell') - \text{dbf}(T_b, \sigma \ell') = \text{slack}(T_\sigma, \sigma \ell') - \text{dbf}(T_b, \sigma \ell'). \quad (17)$$

From Lemma 3.4 we have

$$\text{slack}(T_\sigma, \sigma \ell') = \text{slack}(T_{\text{fill}}, \ell') + (\sigma - 1)\ell' \quad (18)$$

and from Lemma 3.7 we have

$$\text{dbf}(T_b, \sigma \ell') \leq (\sigma - 1)\ell'. \quad (19)$$

By combining Eq. (17), (18) and (19) and using Lemma 3.2, we conclude that

$$\text{slack}(T_c, \sigma \ell') \geq \text{slack}(T_{\text{fill}}, \ell') = \text{slack}(T_1, \ell' \mod \mathcal{P}(T_1)).$$

By assumption, however, $T_1$ is feasible and therefore

$$\text{slack}(T_c, \sigma \ell') \geq \text{slack}(T_1, \ell' \mod \mathcal{P}(T_1)) \geq 0.$$

The lemma follows from this contradiction. ■

We now show the other direction.

Lemma 3.12. If $T_1$ is infeasible, then $T_c$ is also infeasible.

Proof. Assume that $T_1$ is infeasible, then by Corollary 2.3 there exists an $\ell \in \{0, 1, \ldots, \mathcal{P}(T_1) - 1\}$ such that

$$\text{slack}(T_1, \ell) < 0.$$  

By Lemma 3.9, there also exists some $k \in \mathbb{N}$ such that

$$\sigma(k\mathcal{P}(T_1) + \ell) \in B(T_b).$$
By the definition of $B(T_b)$, we therefore have
\[
\text{dbf}(T_b, \sigma(kP(T_1) + \ell)) = (\sigma - 1)(kP(T_1) + \ell).
\] (20)

Note also that from Lemma 3.4 we have
\[
\text{slack}(T_\sigma, \sigma(kP(T_1) + \ell))
= \text{slack}(T_{\text{fill}}, kP(T_1) + \ell) + (\sigma - 1)(kP(T_1) + \ell).
\] (21)

Using Eq. (20) and (21) and Lemma 3.2 we conclude that
\[
\text{slack}(T_c, \sigma(kP(T_1) + \ell))
= \text{slack}(T_\sigma, \sigma(kP(T_1) + \ell)) - \text{dbf}(T_b, \sigma(kP(T_1) + \ell))
= \text{slack}(T_{\text{fill}}, kP(T_1) + \ell)
= \text{slack}(T_1, \ell)
< 0,
\]
and hence that $T_c$ is infeasible by Corollary 2.3. \hfill ■

Finally, our main Theorem follows.

**Theorem 3.13.** The $c$-FEASIBILITY problem is (weakly) coNP-complete for any constant $c$ such that $0 < c < 1$.

**Proof.** First, note that we have described a transformation from FEASIBILITY to $c$-FEASIBILITY, which produces correct outputs according to Lemmas 3.10, 3.11 and 3.12. Secondly, note that the constructed task system $T_c$ has only polynomially many tasks,
\[
|T_1| + 1 + \lceil \log_2(\mathcal{P}(T_1)) \rceil
\]
at most. Also note that the largest parameter in $T_c$ is $p_{\beta-1}$, which has a value of
\[
(\sigma\mathcal{P}(T_1) + 2)^{\lceil \log_2(\mathcal{P}(T_1)) \rceil} \sigma,
\]
which requires only polynomial space. The transformation is then trivially computable in polynomial time.

From Theorem 1.3 we know that FEASIBILITY is coNP-complete. The polynomial transformation therefore proves that $c$-FEASIBILITY is coNP-hard. Because $c$-FEASIBILITY is a special case of FEASIBILITY, we also know that $c$-FEASIBILITY is in coNP. The coNP-completeness of $c$-FEASIBILITY follows. \hfill ■
4 Conclusions

The sporadic task model is one of the basic formalisms for specifying real-time workload. The problem of deciding whether a given sporadic task system is schedulable on a preemptive uniprocessor is of fundamental importance for real-time scheduling theory. The special case that is limited to task systems with utilization bounded by some constant is widely encountered in the literature, thanks to the existence of an algorithm that solves that problem in pseudo-polynomial time. A long-standing open question has been whether this special case also has a polynomial-time solution. We have shown that for any reasonable choice of the constant, the problem is coNP-complete and therefore cannot be solved in polynomial time, assuming $P \neq NP$.

References


