Schedulability Analysis of a Graph-Based Task Model for Mixed-Criticality Systems

Pontus Ekberg · **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.

Keywords Real-time · Mixed-criticality · Task graphs · Schedulability analysis

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. 2011a,b; 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

P. Ekberg · W. Yi
Uppsala University, Department of Information Technology, Box 337, SE-751 05 Uppsala, Sweden
E-mail: pontus.ekberg@it.uu.se

W. Yi
E-mail: yi@it.uu.se

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 schedulabilty 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), which described 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

[1] It is equal assuming the same heuristics are applied in a preprocessing tuning phase.

Baruah et al. (2011c) 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 (2010); Guan et al. (2011); Baruah et al. (2011a, 2012, 2011b); Ekberg and Yi (2012). This list is by no means exhaustive.

EDF-based scheduling of mixed-criticality sporadic task systems was investigated by Baruah et al. (2011a) 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., Su and Zhu 2013; 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 $(V(\tau), E_{\mathrm{cf}}(\tau), E_{\mathrm{ms}}(\tau))$, defined as follows.

- $V(\tau)$ is a set of vertices, representing *job types*.
- Each vertex $v \in V(\tau)$ is labeled with a triple of parameters $(e(v), d(v), \mu(v)) \in \mathbb{N}_{\geqslant 0} \times \mathbb{N}_{\geqslant 0} \times M(T)$, representing worst-case execution time, relative deadline and mode of the corresponding job type, respectively.
- $E_{\mathrm{cf}}(\tau)$ is a set of directed edges representing possible task control flow, such that $\mu(u) = \mu(v)$ for each $(u,v) \in E_{\mathrm{cf}}(\tau)$. In the figures these edges are drawn as straight arrows.
- Each edge $(u,v) \in E_{\mathrm{cf}}(\tau)$ is labeled with a minimum inter-release separation delay parameter $p(u,v) \in \mathbb{N}_{\geqslant 0}$.
- $E_{\mathrm{ms}}(\tau)$ is a set of directed edges representing possible mode switches, such that $\mu(u) \neq \mu(v)$ for each $(u,v) \in E_{\mathrm{ms}}(\tau)$. 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 $(u,v) \in E_{\mathrm{cf}}(\tau)$ we have $d(u) \leqslant p(u,v)$.

Note that, by the above definition, $E_{\mathrm{cf}}(\tau)$ and $E_{\mathrm{ms}}(\tau)$ are disjoint sets. Also, $(V(\tau), E_{\mathrm{cf}}(\tau))$ is a directed graph with disjoint subgraphs for each mode of the task, and $(V(\tau), E_{\mathrm{ms}}(\tau))$ is a directed multipartite graph (colorable with one color per mode). For convenience, let the subgraph in $(V(\tau), E_{\mathrm{cf}}(\tau))$ corresponding to mode μ_i be denoted $\mathrm{DRT}_{\mu_i}(\tau) \overset{\mathrm{def}}{=} (\{v \in V(\tau) \mid \mu(v) = \mu_i\}, \{(u,v) \in E_{\mathrm{cf}}(\tau) \mid \mu(u) = \mu_i\})$.

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 μ_i, an MS-DRT task τ behaves as an ordinary DRT task with graph $\mathrm{DRT}_{\mu_i}(\tau)$. That is, it releases a sequence of jobs that corresponds to some path (represented as a sequence of vertices) in $\mathrm{DRT}_{\mu_i}(\tau)$, such that every vertex on the path matches one released job. More formally, a job of τ 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 τ in μ_i if there is a path $(\pi_1, \pi_2, \ldots)$ through $\mathrm{DRT}_{\mu_i}(\tau)$ such that for all n

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

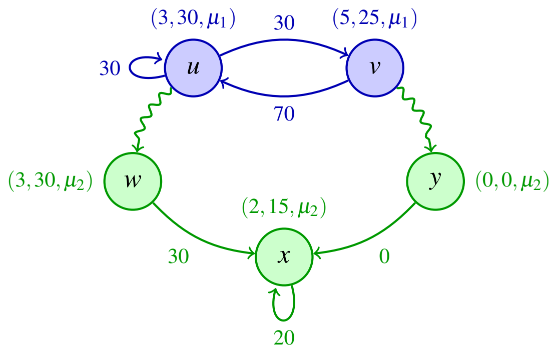


Fig. 1 An example task. The colors help reading, but carry no semantic information. In mode μ_1, this task behaves as a simple two-vertex DRT task, releasing jobs at vertices u and v in some pattern. In μ_2, the behavior is mostly sporadic with repeated job releases at x. The behavior at a mode switch from μ_1 to μ_2 depends on the state. If the latest job released in μ_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 μ_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.

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 τ in μ_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 μ_j to a new mode μ_i. A mode switch from μ_j to μ_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 τ can go on to release a new sequence of jobs $[(r'_1,v'_1),(r'_2,v'_2),\ldots]$ in the new mode μ_i, as long as that sequence prepended by the updated job (r,v) can be generated by τ in μ_i. By these semantics, minimum inter-release separation delays hold across mode switches. In other words, if the latest released job of τ (active or not) was released at time r in a previous mode, then the first control-flow edge $(v,w) \in E_{\mathrm{cf}}(\tau)$ 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 1 (Schedulability) *A mode $\mu_i \in M(T)$ is $\mathcal{A}$-schedulable if all jobs have finished latest at their deadlines while the system is in μ_i and the jobs are executed in μ_i by scheduling algorithm $\mathcal{A}$.*

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.

Example 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.

τ_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).

τ_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 between 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

[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.

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.

τ_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.

τ_4 is a direct extension from a simple two-vertex DRT task to a high-criticality task with different execution-time estimates at the different criticality levels.

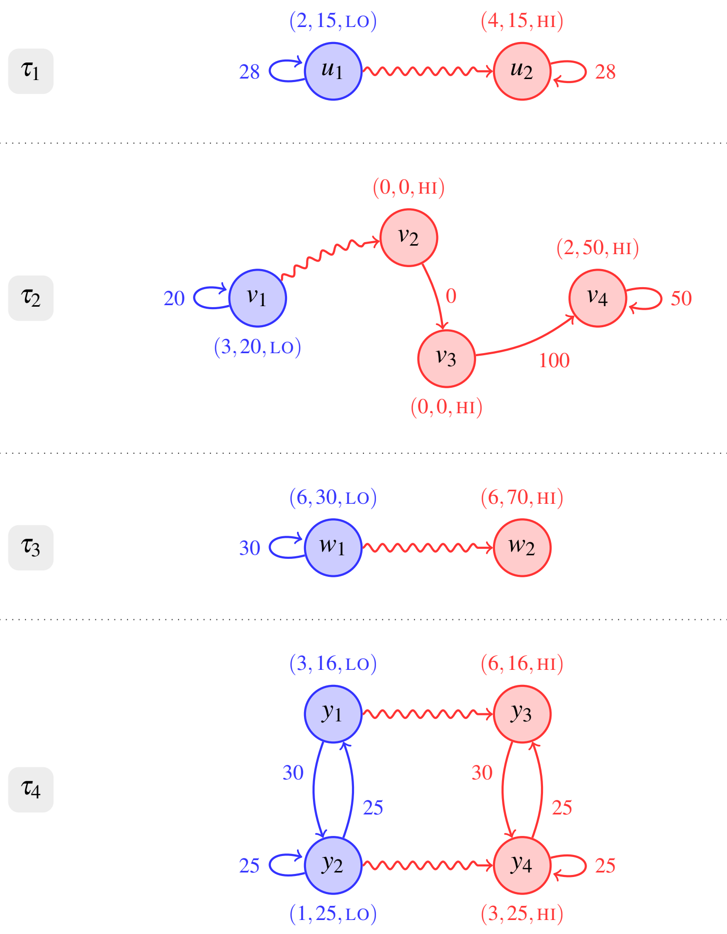


Fig. 2 Example tasks that somewhat similar to ordinary dual-criticality tasks.

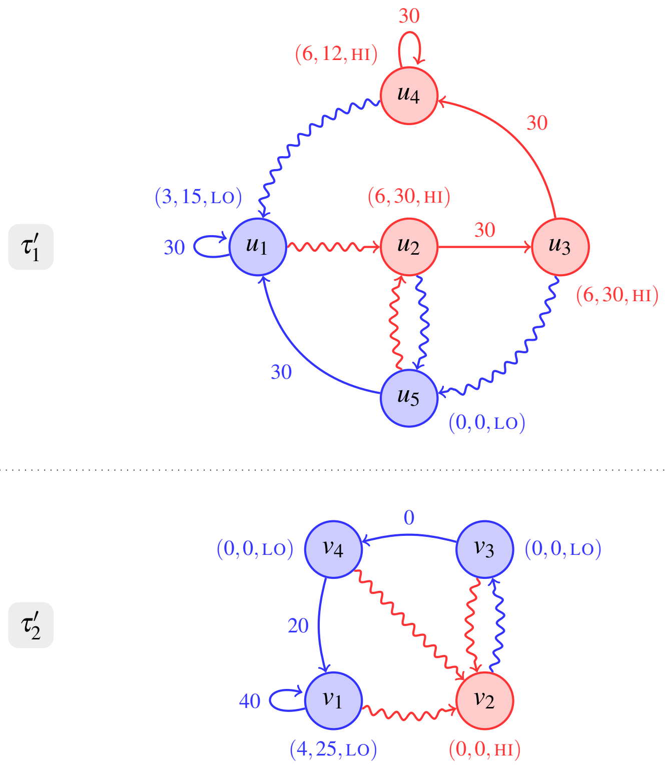


Fig. 3 Example tasks that can switch back to previous modes.

Example 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.

τ'_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.[3]
The intended interpretation is that τ'_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

[3] Some 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. 2011a, 2012; Ekberg and Yi 2012, 2014; Easwaran 2013; Zhang et al. 2014). Automatic deadline tuning is discussed further in Section 4.

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 τ_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.

τ_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 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 2 (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 $(u,v) \in E_{\text{ms}}(\tau)$ and $\mu(u) = \mu_j$ and $\mu(v) = \mu_i$. Also, let the set of immediate predecessor modes to any mode μ_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.

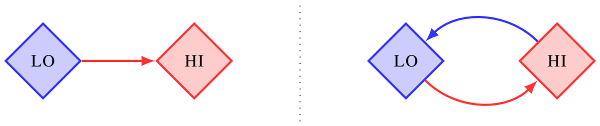


Fig. 4 Mode structures of the tasks in Example 1 (left) and Example 2 (right).

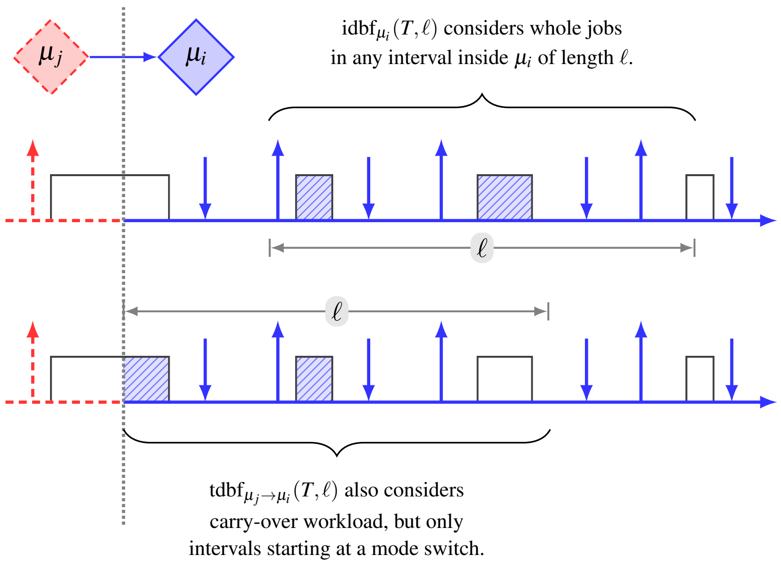

μ_j
μ_i
$\mathrm{idbf}_{\mu_i}(T,\ell)$ considers whole jobs in any interval inside μ_i of length ℓ.
ℓ
ℓ
$\mathrm{tdbf}_{\mu_j\to\mu_i}(T,\ell)$ also considers carry-over workload, but only intervals starting at a mode switch.

Fig. 5 Illustration of internal and transitional demand bound functions.

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 (Internal demand bound functions) *An* internal demand bound function $\mathrm{idbf}_{\mu_i}(T,\ell)$ *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 ℓ, during which the system is continuously in mode μ_i. The top of Figure 5 illustrates this type of demand bound function.*

Definition 4 (Transitional demand bound functions) *A* transitional demand bound function $\mathrm{tdbf}_{\mu_j\to\mu_i}(T,\ell)$ *gives the maximum cumulative execution requirement of jobs from tasks in T in any time interval of length ℓ, such that the interval starts at a mode switch from μ_j to μ_i and during which the system is continuously in mode μ_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 μ_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.

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 $\mathrm{tdbf}_{\mu_j \to \mu_i}(T, \ell)$ we would have to consider, in great detail, the behavior of the system in μ_j, which in turn can depend on the mode preceding μ_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 μ_i, considering a mode switch from another mode μ_j.

$$\mathcal{S}_{\mathrm{EDF}}(T, \mu_i) \overset{\mathrm{def}}{=} \forall \ell \geqslant 0, \quad \mathrm{idbf}_{\mu_i}(T, \ell) \leqslant \ell \tag{1}$$

$$\mathcal{S}_{\mathrm{EDF}}(T, \mu_j \to \mu_i) \overset{\mathrm{def}}{=} \forall \ell \geqslant 0, \quad \mathrm{tdbf}_{\mu_j \to \mu_i}(T, \ell) \leqslant \ell \tag{2}$$

Lemma 1 *Mode μ_i of MS-DRT task system T is EDF-schedulable when either of the following hold.*

1. *Mode μ_i is the first mode the system is in and*

$$\mathcal{S}_{\mathrm{EDF}}(T, \mu_i).$$

2. *Mode μ_i is switched to from μ_j and*

$$\mathcal{S}_{\mathrm{EDF}}(T, \mu_i) \quad \wedge \quad \mathcal{S}_{\mathrm{EDF}}(T, \mu_j \to \mu_i).$$

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

First case: $t_{\mathrm{switch}} \notin [t_{\mathrm{start}}, t_{\mathrm{miss}}]$. In this case the busy period must be entirely in mode μ_i. By the definitions of the busy period and EDF, jobs with absolute deadline latest at t_{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 $\mathrm{idbf}_{\mu_i}(T, \ell)$ is the maximum cumulative execution requirement of jobs from tasks in T that are both released and have deadline in any interval of length ℓ in μ_i, predicate $\mathcal{S}_{\mathrm{EDF}}(T, \mu_i)$ can not hold for $\ell = t_{\mathrm{miss}} - t_{\mathrm{start}}$.

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

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

Corollary 1 *Mode μ_i of MS-DRT task system T is EDF-schedulable if*

$$\mathcal{S}_{\text{EDF}}(T,\mu_i) \quad \wedge \quad \forall \mu_j \in \text{pred}_{G(T)}(\mu_i),\ \mathcal{S}_{\text{EDF}}(T,\mu_j \to \mu_i).$$

$\square$

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 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 τ while in any single mode μ_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 π is a path, let π_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.

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

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

The pair $\langle \tilde{e}(\pi), \tilde{d}(\pi) \rangle$ is called a *demand pair* for path π. 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 τ (see Stigge et al. 2011):

$$\text{idbf}^{\star}_{\mu_i}(\tau,\ell) \overset{\text{def}}{=} \max\{\tilde{e}(\pi) \mid \pi \in \Pi_{\mu_i}(\tau) \wedge \tilde{d}(\pi) \leqslant \ell\} \tag{5}$$

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 exactly matches the internal demand bound function for the task set, as defined in Definition 3.

$$\text{idbf}_{\mu_i}(T,\ell) = \sum_{\tau \in T} \text{idbf}^{\star}_{\mu_i}(\tau,\ell) \tag{6}$$

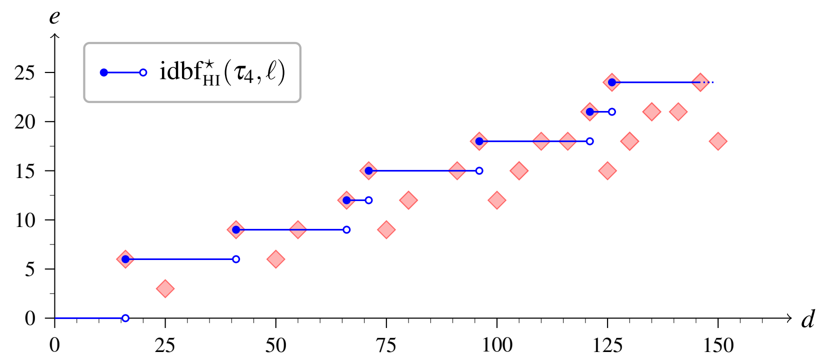


Fig. 6 The (beginning of) function $\mathrm{idbf}^{\star}_{\mathrm{HI}}(\tau_4, \ell)$ for τ_4 from Example 1. The demand pairs for the paths through $\mathrm{DRT}_{\mathrm{HI}}(\tau_4)$ are drawn as diamond-shaped points.

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 2.

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 a transitional demand bound function $\text{tdbf}_{\mu_j \to \mu_i}(T, \ell)$. Assume in the following that the preceding mode μ_j is scheduled by algorithm $\mathcal{A}$ and is $\mathcal{A}$-schedulable, meaning that no deadline can be missed in μ_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 μ_j, and this is the job that was most recently released by the task. Each task τ can then have at most one carry-over job, and if τ switched mode via edge $(u,v) \in E_{\text{ms}}(\tau)$, 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 μ_i.

For each task τ, we must characterize the execution requirements of the (at most) single carry-over job. Let

$$E_{\mu_j \to \mu_i}(\tau) \overset{\text{def}}{=} \left\{ (u,v) \in E_{\text{ms}}(\tau) \mid \mu(u) = \mu_j \wedge \mu(v) = \mu_i \right\}$$

denote the set of mode-switch edges that can take τ from μ_j to μ_i, and let

$$\text{first}_{\mu_j \to \mu_i}(\tau) \overset{\text{def}}{=} \left\{ w \mid (u,v) \in E_{\mu_j \to \mu_i}(\tau) \wedge (v,w) \in E_{\text{cf}}(\tau) \right\}$$

denote the first vertices that can be visited in μ_i via a control-flow edge after such a mode switch.

Clearly, each of the edges $(u,v) \in E_{\mu_j \to \mu_i}(\tau)$ that τ 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 μ_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 μ_j. For a carry-over job at edge $(u,v) \in E_{\mu_j \to \mu_i}(\tau)$, let x be the length of the time interval between the mode switch and the job's absolute deadline in μ_j, as in Figure 7. We know that $x \in [0, d(u)]$. Further, because μ_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 μ_j. The job's remaining execution time budget in μ_j at the time of the mode switch can therefore not exceed the length of the time interval until its deadline in μ_j, and is then at most $\min(e(u), x)$. In the new mode μ_i,

[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 μ_i.

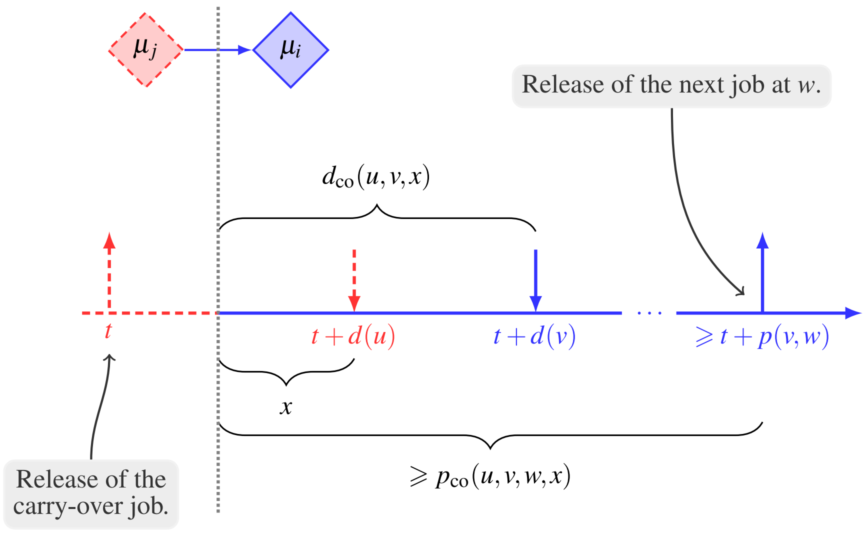

μ_j
μ_i
Release of the next job at w.
$d_{\text{co}}(u,v,x)$
t
$t+d(u)$
$t+d(v)$
$\geqslant t+p(v,w)$
x
$\geqslant p_{\text{co}}(u,v,w,x)$
Release of the carry-over job.

Fig. 7 A carry-over job of τ at a mode switch via $(u,v) \in E_{\mu_j \to \mu_i}(\tau)$. The first job to be released inside μ_i is at vertex $w \in \text{first}_{\mu_j \to \mu_i}(\tau)$.

the total budget is changed to $e(v)$ and the most that can remain of it immediately after the mode switch is

$$e_{\text{co}}(u,v,x) \overset{\text{def}}{=} \max\left(0,\ e(v) - e(u) + \min(e(u),x)\right). \tag{7}$$

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 μ_i is latest at the end of the interval, as

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

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 μ_i, is

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

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

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 τ. In this case, all the jobs from τ that can add to the cumulative execution requirement of

$\mathrm{tdbf}_{\mu_j\to\mu_i}(T,\ell)$ are both released and have deadline in an interval of length ℓ 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

$$\mathrm{pair}_{\mathrm{nco}}(\pi) \overset{\text{def}}{=} \langle \tilde{e}(\pi), \tilde{d}(\pi) \rangle.$$

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

$$\mathrm{nco}_{\mu_j\to\mu_i}(\tau) \overset{\text{def}}{=} \{\mathrm{pair}_{\mathrm{nco}}(\pi) \mid \pi \in \mathrm{Paths}\}, \tag{10}$$

where $\mathrm{Paths} = \{\pi \in \Pi_{\mu_i}(\tau) \mid \pi_1 \in \mathrm{first}_{\mu_j\to\mu_i}(\tau) \vee |\pi| = 0\}$.

We now look at the case where there is a carry-over job from task τ. Each job sequence to consider from τ in this case corresponds to some path $\pi \in \Pi_{\mu_i}(\tau)$ where the first vertex is the carry-over job, i.e., where $(u,\pi_1) \in E_{\mu_j\to\mu_i}(\tau)$ 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 μ_j. Given such an $x \in [0, d(u)]$, the cumulative execution requirement to consider for the entire job sequence is therefore at most

$$\tilde{e}_{\mathrm{co}}(u,\pi,x) \overset{\text{def}}{=} e_{\mathrm{co}}(u,\pi_1,x) + \tilde{e}(\pi_{2\cdots|\pi|}).$$

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 μ_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

$$\tilde{d}_{\mathrm{co}}(u,\pi,x) \overset{\text{def}}{=} \begin{cases} d_{\mathrm{co}}(u,\pi_1,x) & \text{if } |\pi| = 1, \\ p_{\mathrm{co}}(u,\pi_1,\pi_2,x) + \tilde{d}(\pi_{2\cdots|\pi|}) & \text{otherwise.} \end{cases}$$

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

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

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

$$\mathrm{co}_{\mu_j\to\mu_i}(\tau) \overset{\text{def}}{=} \{\mathrm{pair}_{\mathrm{co}}(u,\pi,x) \mid (u,\pi,x) \in \mathrm{Vals}\}, \tag{11}$$

where we have $\mathrm{Vals} = \{(u,\pi,x) \mid \pi \in \Pi_{\mu_i}(\tau) \wedge (u,\pi_1) \in E_{\mu_j\to\mu_i}(\tau) \wedge x \in [0, d(u)]\}$.

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 τ in the same manner as in Eq. (5).[5]

$$\mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell) \overset{\mathrm{def}}{=} \max\{e \mid \langle e, d\rangle \in \mathrm{Pairs} \wedge d \leqslant \ell\}, \tag{12}$$

where $\mathrm{Pairs} = \mathrm{nco}_{\mu_j \to \mu_i}(\tau) \bigcup \mathrm{co}_{\mu_j \to \mu_i}(\tau)$.

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 2 *If mode μ_j is scheduled by algorithm $\mathcal{A}$ and is $\mathcal{A}$-schedulable, then*

$$\forall \ell \geqslant 0, \quad \sum_{\tau \in T} \mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell) \geqslant \mathrm{tdbf}_{\mu_j \to \mu_i}(T, \ell).$$

Proof By the reasoning above. □

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 2 it was shown that $\mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell)$ 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 $\mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau, \ell)$ 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 $\mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell)$. 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 demand pairs in Eq. (11):

$$\mathrm{co}^{\star}_{\mu_j \to \mu_i}(\tau) \overset{\mathrm{def}}{=} \{\mathrm{pair}_{\mathrm{co}}(u, \pi, e(u)) \mid (u, \pi) \in \mathrm{Vals}^{\star}\}, \tag{13}$$

where $\mathrm{Vals}^{\star} = \{(u, \pi) \mid \pi \in \Pi_{\mu_i}(\tau) \wedge (u, \pi_1) \in E_{\mu_j \to \mu_i}(\tau)\}$.

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

$$\mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau, \ell) \overset{\mathrm{def}}{=} \max\{e \mid \langle e, d\rangle \in \mathrm{Pairs}^{\star} \wedge d \leqslant \ell\}, \tag{14}$$

where $\mathrm{Pairs}^{\star} = \mathrm{nco}_{\mu_j \to \mu_i}(\tau) \bigcup \mathrm{co}^{\star}_{\mu_j \to \mu_i}(\tau)$.

[5] Here it can be noted that if we would model a sporadic mixed-criticality task with MS-DRT, such as task τ_1 in Example 1, the function $\mathrm{tdbf}^{\mathrm{ub}}_{\mathrm{LO} \to \mathrm{HI}}(\tau_1, \ell)$ would be equal to function $\mathrm{dbf}_{\mathrm{LO,HI}}(\tau_1, \ell)$ from Eq. (8) in Ekberg and Yi (2014), although the formulation is completely different.

The following lemma shows that $\mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau, \ell)$ preserves safety in schedulability analysis.

Lemma 3

$$\begin{aligned} &\exists \ell_1 \geqslant 0, \quad \sum_{\tau \in T} \mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell_1) > \ell_1 \quad \iff \\ &\exists \ell_2 \geqslant 0, \quad \sum_{\tau \in T} \mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau, \ell_2) > \ell_2 \end{aligned}$$

Proof The proof is in Appendix C. □

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

$$\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_i) \stackrel{\text{def}}{=} \forall \ell \geqslant 0, \sum_{\tau \in T} \mathrm{idbf}^{\star}_{\mu_i}(\tau, \ell) \leqslant \ell \tag{15}$$

$$\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_j \to \mu_i) \stackrel{\text{def}}{=} \forall \ell \geqslant 0, \sum_{\tau \in T} \mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau, \ell) \leqslant \ell \tag{16}$$

These predicates are safe replacements for their original counterparts:

Lemma 4 *First, $\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_i) \iff \mathcal{S}_{\mathrm{EDF}}(T, \mu_i)$. Second, if mode μ_j is scheduled by algorithm $\mathcal{A}$ and is $\mathcal{A}$-schedulable, then*

$$\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_j \to \mu_i) \Longrightarrow \mathcal{S}_{\mathrm{EDF}}(T, \mu_j \to \mu_i).$$

Proof From Eq. (6) we directly get $\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_i) \iff \mathcal{S}_{\mathrm{EDF}}(T, \mu_i)$. Lemmas 2 and 3 taken together give us $\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_j \to \mu_i) \Longrightarrow \mathcal{S}_{\mathrm{EDF}}(T, \mu_j \to \mu_i)$. □

Our two main theorems follow. The first is about the EDF-schedulability of one mode given the schedulability of the possible preceding modes.

Theorem 1 *Mode μ_i of MS-DRT task system T is EDF-schedulable if each mode $\mu_j \in \mathrm{pred}_{G(T)}(\mu_i)$ is scheduled by algorithm $\mathcal{A}_j$ and is $\mathcal{A}_j$-schedulable and if*

$$\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_i) \quad \wedge \quad \forall \mu_j \in \mathrm{pred}_{G(T)}(\mu_i), \, \mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_j \to \mu_i).$$

Proof By Corollary 1 and Lemma 4. □

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

$$\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_i) \quad \wedge \quad \forall \mu_j \in \mathrm{pred}_{G(T)}(\mu_i), \, \mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_j \to \mu_i).$$

Proof By Lemmas 1 and 4 and by induction on the sequence of modes the system transitions through. The base case is the first mode μ_i that the system is in, which is guaranteed schedulability by $\mathcal{S}^{\star}_{\mathrm{EDF}}(T, \mu_i)$. □

3.4 Efficiently Computing the Demand Bound Functions

We now look at how to evaluate the predicates $\mathcal{S}^{\star}_{\text{EDF}}(T,\,\mu_i)$ and $\mathcal{S}^{\star}_{\text{EDF}}(T,\,\mu_j \to \mu_i)$ efficiently. There are two major challenges:

1. We cannot actually compute the functions $\text{idbf}^{\star}_{\mu_i}(\tau,\ell)$ and $\text{tdbf}^{\star}_{\mu_j\to\mu_i}(\tau,\ell)$ at *all* values of $\ell \geqslant 0$ because of the infinite domain.
2. Even for a given value of ℓ, the number of possible paths through the graphs is generally exponential in ℓ.

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 ℓ 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 $\text{idbf}^{\star}_{\mu_i}(\tau,\ell)$ 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 $\{\text{DRT}_{\mu_i}(\tau) \mid \tau \in T\}$, and can thus be used to evaluate $\mathcal{S}^{\star}_{\text{EDF}}(T,\,\mu_i)$.

Mapping of functions to DRT tasks For $\text{tdbf}^{\star}_{\mu_j\to\mu_i}(\tau,\ell)$ 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 $\text{tdbf}^{\star}_{\mu_j\to\mu_i}(\tau,\ell)$. This allows us to reuse the existing methods without the need to reprove their correctness in a new setting.[6]

The key observation that makes the construction possible is that for each mode-switch edge $(u,v) \in E_{\mu_j\to\mu_i}(\tau)$, we have reduced all the possible carry-over jobs considered for $\text{tdbf}^{\star}_{\mu_j\to\mu_i}(\tau,\ell)$ to the equivalent of a single concrete job. The idea is to construct a new DRT task $\text{DRT}_{\mu_j\to\mu_i}(\tau)$ by taking the graph $\text{DRT}_{\mu_i}(\tau)$ 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 $(u,v) \in E_{\mu_j\to\mu_i}(\tau)$, we add a vertex named $[\![u,v]\!]$ after the corresponding edge to $\text{DRT}_{\mu_j\to\mu_i}(\tau)$, and label it with $(e_{\text{co}}(u,v,e(u)), d_{\text{co}}(u,v,e(u)))$.
2. For every newly added vertex $[\![u,v]\!]$ and edge $(v,w) \in E_{\text{cf}}(\tau)$, we add an edge $([\![u,v]\!],w)$ and label it with $p_{\text{co}}(u,v,w,e(u))$.

It is evident that for every $(u,\pi) \in \text{Vals}^{\star}$ used in Eq. (13), there is now a path π' in $\text{DRT}_{\mu_j\to\mu_i}(\tau)$ such that $\pi'_1 = [\![u,\pi_1]\!]$ and $\pi'_{2\cdots|\pi'|} = \pi_{2\cdots|\pi|}$. By construction, we have $\langle \tilde{e}(\pi'), \tilde{d}(\pi') \rangle = \text{pair}_{\text{co}}(u,\pi,e(u))$.

[6] 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 an 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 set of demand pairs for those paths in $\textsc{drt}_{\mu_j \to \mu_i}(\tau)$ that start at one of the newly added vertices therefore equal the set $\mathrm{co}^{\star}_{\mu_j \to \mu_i}(\tau)$. The demand pairs in $\mathrm{nco}_{\mu_j \to \mu_i}(\tau)$ still correspond to the paths starting at one of the vertices in $\mathrm{first}_{\mu_j \to \mu_i}(\tau)$ (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 $\{\textsc{drt}_{\mu_j \to \mu_i}(\tau) \mid \tau \in T\}$, but restrict the considered paths to those starting at either one of the new vertices or a vertex in $\mathrm{first}_{\mu_j \to \mu_i}(\tau)$. It follows that the demand bound function computed in this way for $\textsc{drt}_{\mu_j \to \mu_i}(\tau)$ equal the function $\mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau, \ell)$ over ℓ.

As an illustration, Figure 8 shows the DRT tasks constructed from the tasks in Example 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 $\textsc{drt}_{\textsc{lo} \to \textsc{hi}}(\tau_2')$), but this is not a problem because the dynamic programming graph traversal algorithm will ignore such vertices almost immediately.

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 $\textsc{drt}_{\mu_j \to \mu_i}(\tau)$ has the same utilization as $\textsc{drt}_{\mu_i}(\tau)$ because the added vertices are never part of a cycle. The evaluation of $\mathcal{S}^{\star}_{\textsc{edf}}(T, \mu_i)$ or $\mathcal{S}^{\star}_{\textsc{edf}}(T, \mu_j \to \mu_i)$ is therefore of pseudo-polynomial time complexity, following the results of Stigge et al. (2011), as long as the asymptotic utilization in μ_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 2, we need to perform $|V| + |E|$ such pseudo-polynomial time procedures, where $(V, E) = G(T)$. Note that traversals in $\textsc{drt}_{\mu_j \to \mu_i}(\tau)$ are equivalent to traversals in $\textsc{drt}_{\mu_i}(\tau)$ after the first few vertices because the added vertices in $\textsc{drt}_{\mu_j \to \mu_i}(\tau)$ have no incoming edges. Much of the computation needed for establishing schedulability of a given mode μ_i (i.e., evaluating $\mathcal{S}^{\star}_{\textsc{edf}}(T, \mu_i)$ and each $\mathcal{S}^{\star}_{\textsc{edf}}(T, \mu_j \to \mu_i)$) therefore consists of repeated graph traversals in $\textsc{drt}_{\mu_i}(\tau)$ 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. 2011a, 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.

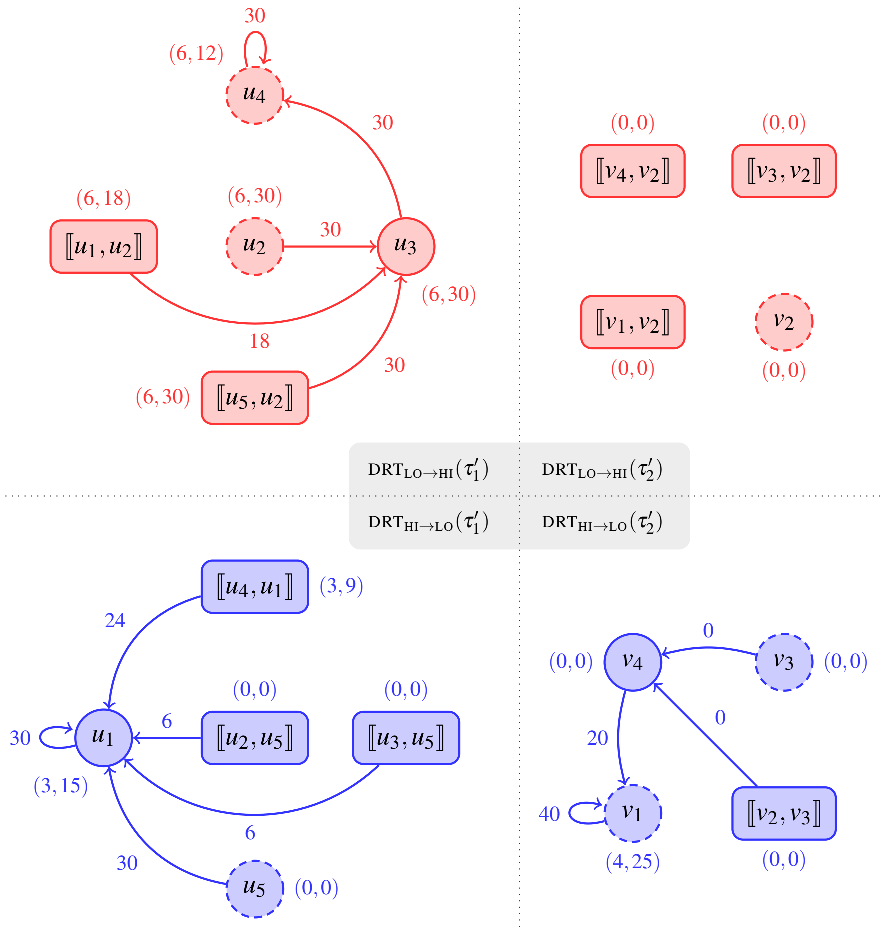


Fig. 8 The DRT tasks that are generated from the two MS-DRT tasks in Example 2. Valid start vertices are those drawn with solid lines; no path starting at a dashed vertex is considered when computing their demand bound functions.

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 τ has a carry-over job at some mode-switch edge $(u, v) \in E_{\mu_j \to \mu_i}(\tau)$. The carry-over job's timing properties in mode μ_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 δ, then $e_{\mathrm{co}}(u, v, x)$ would remain the same (for a given x) while $d_{\mathrm{co}}(u, v, x)$ and $p_{\mathrm{co}}(u, v, w, x)$ would both increase by up to δ. 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 μ_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 2 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 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 μ_i, i.e., $\mathrm{idbf}^{\star}_{\mu_i}(\tau,\ell)$ and $\mathrm{tdbf}^{\star}_{\mu_j \to \mu_i}(\tau,\ell)$ for each predecessor mode μ_j. If jobs of type u can be carried over to another mode μ_k, then the parameter change can also affect $\mathrm{tdbf}^{\star}_{\mu_i \to \mu_k}(\tau,\ell)$. These are demand bound functions defined by the demand pairs of paths in one of the graphs $\mathrm{DRT}_{\mu_i}(\tau)$, $\mathrm{DRT}_{\mu_j \to \mu_i}(\tau)$ or $\mathrm{DRT}_{\mu_i \to \mu_k}(\tau)$.

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 affect 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 $\langle \tilde{e}(\pi), \tilde{d}(\pi) \rangle$ of such a path π is only affected by the change to $d(u)$ if either π_1 is $[\![u,v]\!]$ or $\pi_{|\pi|}$ is u or $[\![v,u]\!]$. Changes to deadlines of intermediate vertices in π do not affect its demand pair. In fact, given any path π we only need to know the identities of vertices π_1, π_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 π_2 is needed to calculate the minimum separation between π_1 and π_2 with Eq. (9).) See Figure 9 for an illustration.

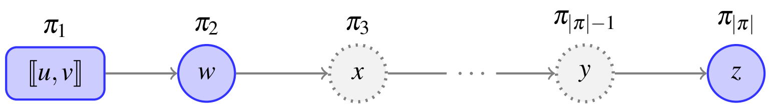


Fig. 9 At most three vertices of path π must be considered when updating its demand pair upon a deadline change. For this path, a decrease of $d(z)$ results in a decrease of $\tilde{d}(\pi)$, and a decrease of $d(u)$ may instead increase $\tilde{d}(\pi)$. Changes to the deadline of other vertices, such 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.

Acknowledgements This work was supported in part by the Swedish Research Council within the UPMARC Linnaeus centre of Excellence. We would like to thank the anonymous reviewers for their helpful comments.

Appendix A: Some Preliminary Experiments

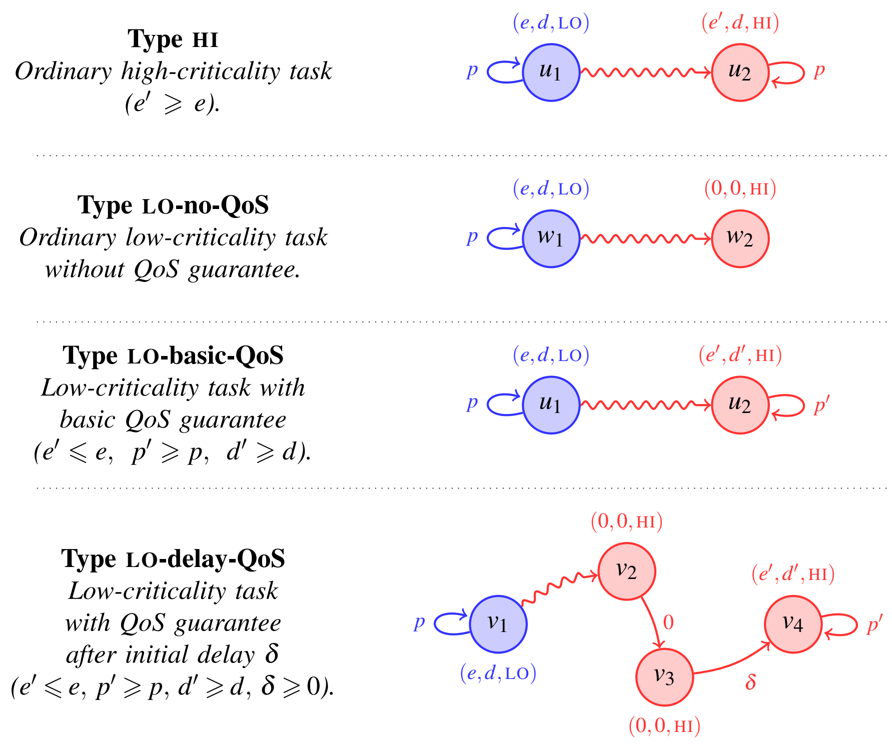
A.1 Motivation

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 τ_2 of Example 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.



Type HI
Ordinary high-criticality task ($e' \geqslant e$).
(e, d, LO)
(e', d, HI)
p
u_1
u_2
p
Type LO-no-QoS
Ordinary low-criticality task without QoS guarantee.
(e, d, LO)
$(0, 0, \text{HI})$
p
w_1
w_2
Type LO-basic-QoS
Low-criticality task with basic QoS guarantee ($e' \leqslant e$, $p' \geqslant p$, $d' \geqslant d$).
(e, d, LO)
(e', d', HI)
p
u_1
u_2
p'
Type LO-delay-QoS
Low-criticality task with QoS guarantee after initial delay δ ($e' \leqslant e$, $p' \geqslant p$, $d' \geqslant d$, $\delta \geqslant 0$).
$(0, 0, \text{HI})$
v_2
(e', d', HI)
p
v_1
0
v_4
p'
(e, d, LO)
v_3
δ
$(0, 0, \text{HI})$

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

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, 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

$Pr_{\text{HI}} = 0.5,$	*the probability of each task to be of high criticality,*
$e^{\text{LO}}_{\text{max}} = 10,$	*the maximum execution time budget in mode* LO,
$p^{\text{LO}}_{\text{max}} = 200,$	*the maximum period in mode* LO,
$e^{\text{HI}}_{\text{fact}} = 4,$	*the maximum factor of execution time increase in* HI *(for* HI*-tasks),*
$e^{\text{LO}}_{\text{fact}} = 0.5,$	*the minimum factor of execution time decrease in* HI *(for* LO*-tasks),*
$p^{\text{LO}}_{\text{fact}} = 2,$	*the maximum factor of period increase in* HI *(for* LO*-tasks).*

The first three constants were chosen somewhat arbitrarily. The value for $e^{\text{HI}}_{\text{fact}}$ 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 ($e^{\text{LO}}_{\text{fact}}$) and double their periods ($p^{\text{LO}}_{\text{fact}}$) 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 $U_{\text{avg}}(T) \in [U^* - 0.005, U^* + 0.005]$, 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 μ_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 δ-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^* 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 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 TuneSystem that only attempts to tune deadlines of high-criticality tasks.

We still have to determine the values of the δ parameters of the tasks of type LO-delay-QoS. For simplicity, we assume that all such tasks in a task set T_{delay} have the same value for their δ parameter. For each such task set we do a binary search on the ordered set $\{0, \ldots, p^{\text{LO}}_{\max}\}$ to find the minimal value for δ with which T_{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 δ 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 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.

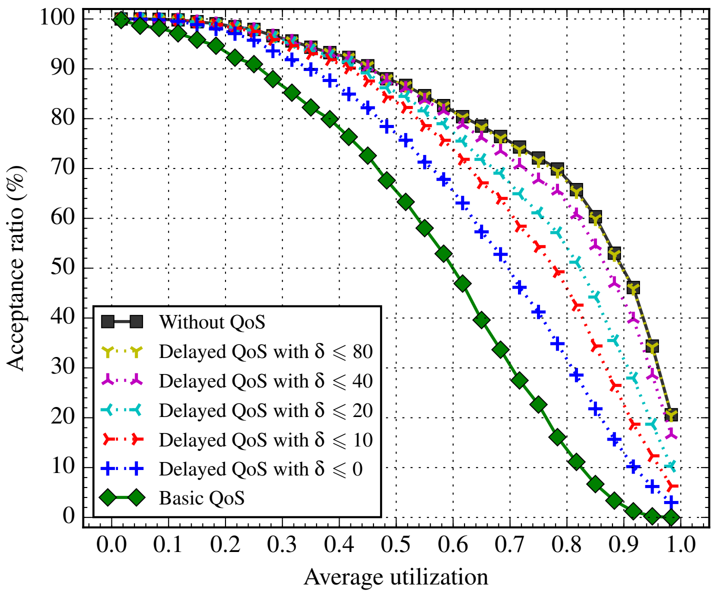


Fig. 11 Acceptance ratios for the three different types of task sets.

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

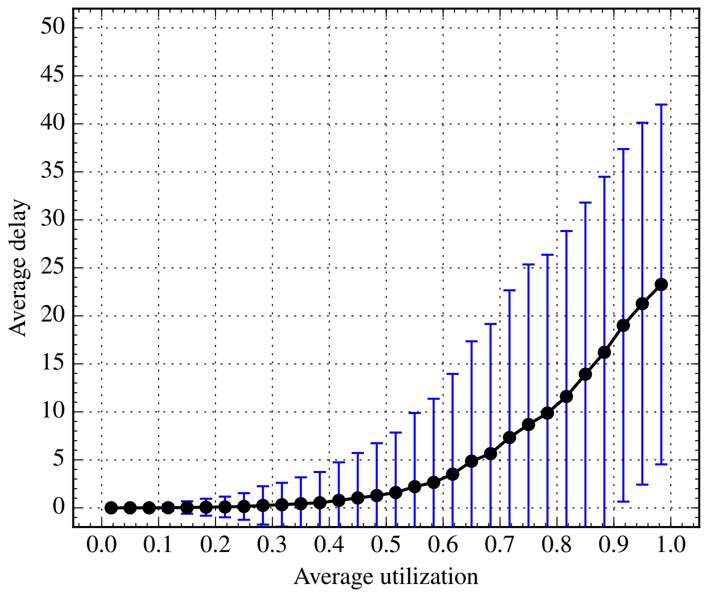


Fig. 12 The average of the minimum values for δ needed to make the task sets schedulable.

Algorithm 1: Algorithm for generating random task sets.

```
Function generate-task-sets($U^*$, $Pr_{\mathrm{HI}}$, $e^{\mathrm{LO}}_{\max}$, $p^{\mathrm{LO}}_{\max}$, $e^{\mathrm{LO}}_{\mathrm{fact}}$, $e^{\mathrm{HI}}_{\mathrm{fact}}$, $p^{\mathrm{LO}}_{\mathrm{fact}}$):
    $T_{\mathrm{no}}, T_{\mathrm{basic}}, T_{\mathrm{delay}} \leftarrow \emptyset, \emptyset, \emptyset$
    repeat
        with probability $Pr_{\mathrm{HI}}$ do
            $(e, d, p, e') \leftarrow$ get-hi-task-params($e^{\mathrm{LO}}_{\max}$, $p^{\mathrm{LO}}_{\max}$, $e^{\mathrm{HI}}_{\mathrm{fact}}$)
            $\tau_{\mathrm{HI}} \leftarrow$ Type HI task with parameters $(e, d, p, e')$
            $T_{\mathrm{no}} \leftarrow T_{\mathrm{no}} \cup \{\tau_{\mathrm{HI}}\}$
            $T_{\mathrm{basic}} \leftarrow T_{\mathrm{basic}} \cup \{\tau_{\mathrm{HI}}\}$
            $T_{\mathrm{delay}} \leftarrow T_{\mathrm{delay}} \cup \{\tau_{\mathrm{HI}}\}$
        otherwise do
            $(e, d, p, e', d', p') \leftarrow$ get-lo-task-params($e^{\mathrm{LO}}_{\max}$, $p^{\mathrm{LO}}_{\max}$, $e^{\mathrm{LO}}_{\mathrm{fact}}$, $p^{\mathrm{LO}}_{\mathrm{fact}}$)
            $\tau^{\mathrm{no}}_{\mathrm{LO}} \leftarrow$ Type LO-no-QoS task with parameters $(e, d, p)$
            $T_{\mathrm{no}} \leftarrow T_{\mathrm{no}} \cup \{\tau^{\mathrm{no}}_{\mathrm{LO}}\}$
            $\tau^{\mathrm{basic}}_{\mathrm{LO}} \leftarrow$ Type LO-basic-QoS task with parameters $(e, d, p, e', d', p')$
            $T_{\mathrm{basic}} \leftarrow T_{\mathrm{basic}} \cup \{\tau^{\mathrm{basic}}_{\mathrm{LO}}\}$
            $\tau^{\mathrm{delay}}_{\mathrm{LO}} \leftarrow$ Type LO-delay-QoS task with parameters $(e, d, p, e', d', p')$
            $T_{\mathrm{delay}} \leftarrow T_{\mathrm{delay}} \cup \{\tau^{\mathrm{delay}}_{\mathrm{LO}}\}$        // Task parameter $\delta$ to be set later
        if $\neg$is-valid-task-set($T_{\mathrm{basic}}$) then
            $T_{\mathrm{no}}, T_{\mathrm{basic}}, T_{\mathrm{delay}} \leftarrow \emptyset, \emptyset, \emptyset$
    until $U^* - 0.005 \leqslant U_{\mathrm{avg}}(T_{\mathrm{basic}}) \leqslant U^* + 0.005$
    return $(T_{\mathrm{no}}, T_{\mathrm{basic}}, T_{\mathrm{delay}})$

Function get-hi-task-params($e^{\mathrm{LO}}_{\max}$, $p^{\mathrm{LO}}_{\max}$, $e^{\mathrm{HI}}_{\mathrm{fact}}$):
    $e \leftarrow$ random sample from $\mathcal{U}\{1, \ldots, e^{\mathrm{LO}}_{\max}\}$    // Discrete uniform distribution
    $e_{\mathrm{fact}} \leftarrow$ random sample from $\mathcal{U}[1, e^{\mathrm{HI}}_{\mathrm{fact}}]$    // Continuous uniform distribution
    $e' \leftarrow \lceil e \cdot e_{\mathrm{fact}} \rceil$
    $p \leftarrow$ random sample from $\mathcal{U}\{e', \ldots, p^{\mathrm{LO}}_{\max}\}$
    $d_{\mathrm{fact}} \leftarrow$ random sample from $\mathcal{U}[0, 1]$
    $d \leftarrow \max(e', \lceil p \cdot d_{\mathrm{fact}} \rceil)$
    return $(e, d, p, e')$

Function get-lo-task-params($e^{\mathrm{LO}}_{\max}$, $p^{\mathrm{LO}}_{\max}$, $e^{\mathrm{LO}}_{\mathrm{fact}}$, $p^{\mathrm{LO}}_{\mathrm{fact}}$):
    $e \leftarrow$ random sample from $\mathcal{U}\{1, \ldots, e^{\mathrm{LO}}_{\max}\}$
    $e_{\mathrm{fact}} \leftarrow$ random sample from $\mathcal{U}[e^{\mathrm{LO}}_{\mathrm{fact}}, 1]$
    $e' \leftarrow \lceil e \cdot e_{\mathrm{fact}} \rceil$
    $p \leftarrow$ random sample from $\mathcal{U}\{e, \ldots, p^{\mathrm{LO}}_{\max}\}$
    $p_{\mathrm{fact}} \leftarrow$ random sample from $\mathcal{U}[1, p^{\mathrm{LO}}_{\mathrm{fact}}]$
    $p' \leftarrow \lceil p \cdot p_{\mathrm{fact}} \rceil$
    $d_{\mathrm{fact}} \leftarrow$ random sample from $\mathcal{U}[0, 1]$
    $d \leftarrow \max(e, \lceil p \cdot d_{\mathrm{fact}} \rceil)$
    $d' \leftarrow \max(e', \lceil p' \cdot d_{\mathrm{fact}} \rceil)$
    return $(e, d, p, e', d', p')$

Function is-valid-task-set($T$):
    if $U(T, \mathrm{LO}) \leqslant 0.99 \wedge U(T, \mathrm{HI}) \leqslant 0.99 \wedge \mathcal{S}_{\mathrm{EDF}}(T, \mathrm{LO}) \wedge \mathcal{S}_{\mathrm{EDF}}(T, \mathrm{HI})$ then
        return true
    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.

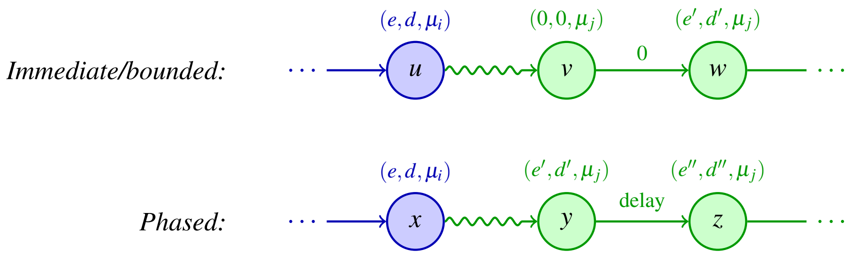


Fig. 13 Simple modeling of transitions for different mode change types.

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.

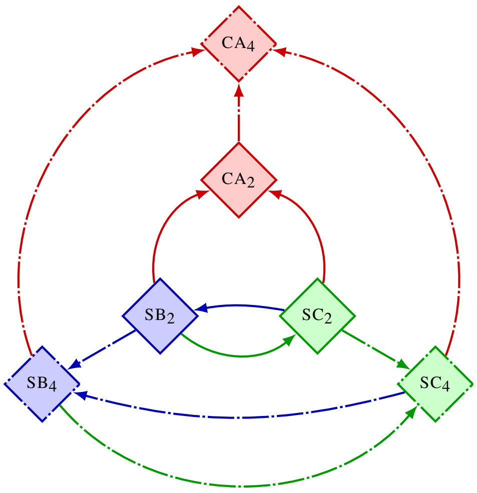


Fig. 14 Mode structure of the cruise-control system.

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

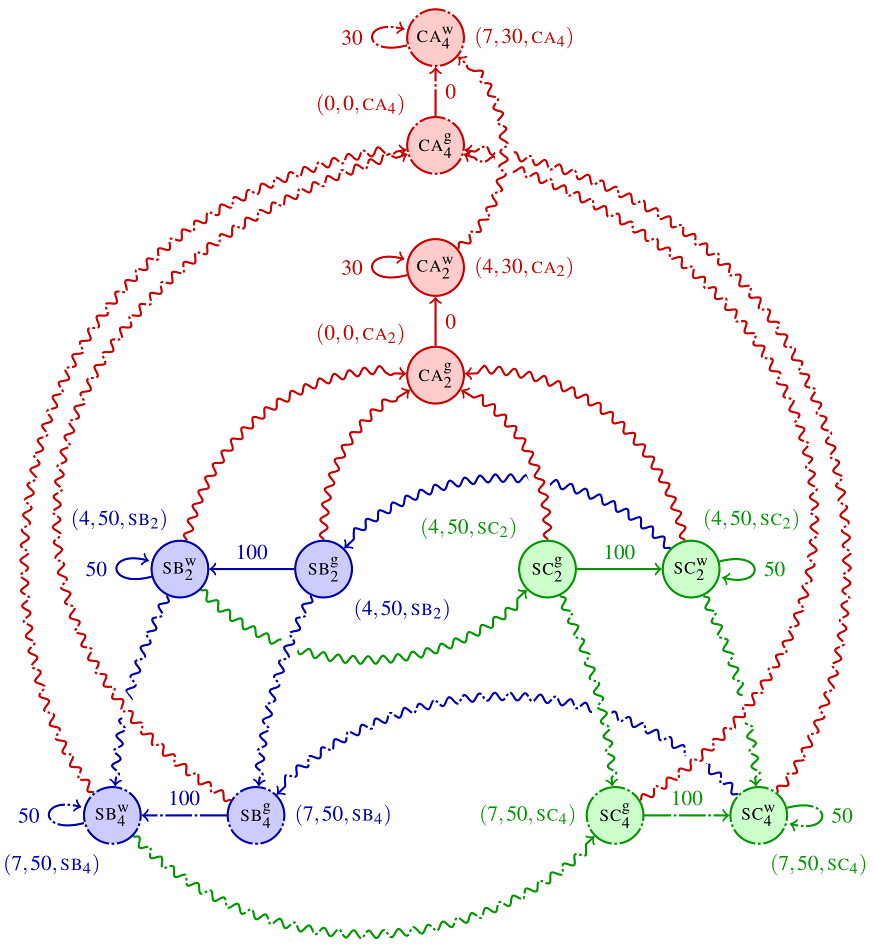


Fig. 15 The proximity analysis task.

mode switches, including complex situations such as a transition to CA2 happening in the middle of a phased transition between SB2 and SC2, closely followed by a transition to CA4. 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 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 SB_2 to SB_4) 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 $\mathrm{CA}_2^{\mathrm{g}}$ to $\mathrm{CA}_4^{\mathrm{g}}$ with the interpretation that no time ever passes before moving on from $\mathrm{CA}_2^{\mathrm{g}}$ to $\mathrm{CA}_2^{\mathrm{w}}$.

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.

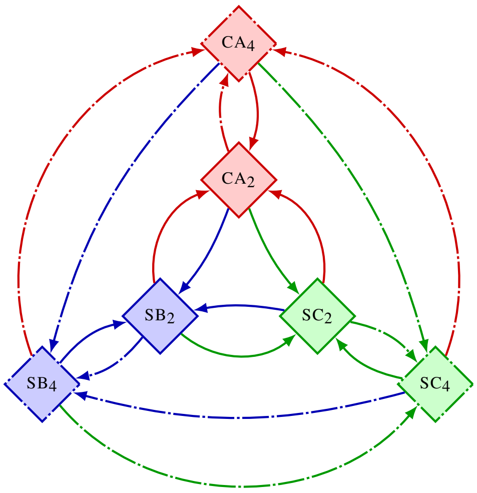


Fig. 16 An extended mode structure that is strongly connected.

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

To prove Lemma 3, we first define a relation on demand pairs and prove an auxiliary lemma.

Definition 5 (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*

$$e \geqslant e' \text{ and } e - e' \geqslant d - d'.$$

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.

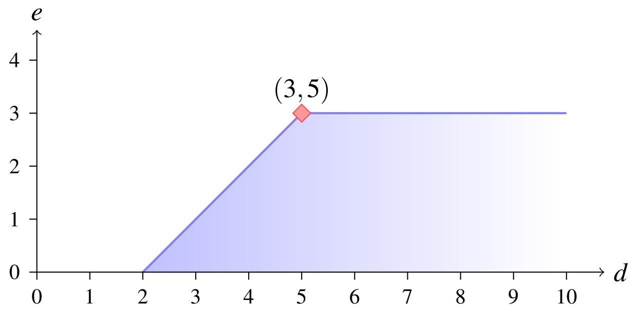


Fig. 17 The demand pair $(3,5)$ covers all pairs in the shaded area.

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

Lemma 5 *For each demand pair $\langle e,d\rangle \in \mathrm{nco}_{\mu_j\to\mu_i}(\tau) \bigcup \mathrm{co}_{\mu_j\to\mu_i}(\tau)$ there exist an $\langle e^\star,d^\star\rangle \in \mathrm{nco}_{\mu_j\to\mu_i}(\tau) \bigcup \mathrm{co}^{\star}_{\mu_j\to\mu_i}(\tau)$ such that $\langle e^\star,d^\star\rangle \trianglerighteq \langle e,d\rangle$.*

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

We consider instead a demand pair $\langle e,d\rangle \in \mathrm{co}_{\mu_j\to\mu_i}(\tau)$. From Eq. (11) it is evident that $\langle e,d\rangle = \mathrm{pair}_{\mathrm{co}}(u,\pi,x)$ for some $(u,\pi,x) \in \mathrm{Vals}$. We split the proof into three cases.

Case 1 ($x \geqslant e(u)$):

Let $\langle e^\star,d^\star\rangle = \mathrm{pair}_{\mathrm{co}}(u,\pi,e(u))$. Clearly, $\langle e^\star,d^\star\rangle \in \mathrm{co}^{\star}_{\mu_j\to\mu_i}(\tau)$. We calculate

$$\begin{aligned} e &= \tilde{e}_{\mathrm{co}}(u,\pi,x) \\ &= e_{\mathrm{co}}(u,\pi_1,x) + \tilde{e}(\pi_{2\cdots|\pi|}) \\ &= e_{\mathrm{co}}(u,\pi_1,e(u)) + \tilde{e}(\pi_{2\cdots|\pi|}) \\ &= \tilde{e}_{\mathrm{co}}(u,\pi,e(u)) \\ &= e^\star \end{aligned}$$

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

Case 2 ($x \leqslant e(u) - e(\pi_1)$):

Let $\langle e^\star, d^\star \rangle = \mathrm{pair}_{\mathrm{nco}}(\pi_{2\cdots|\pi|})$. Because either $|\pi_{2\cdots|\pi|}| = 0$ or $\pi_2 \in \mathrm{first}_{\mu_j \to \mu_i}(\tau)$, we have $\langle e^\star, d^\star \rangle \in \mathrm{nco}_{\mu_j \to \mu_i}(\tau)$. Further,

$$\begin{aligned} e &= \tilde{e}_{\mathrm{co}}(u, \pi, x) \\ &= e_{\mathrm{co}}(u, \pi_1, x) + \tilde{e}(\pi_{2\cdots|\pi|}) \\ &= 0 + \tilde{e}(\pi_{2\cdots|\pi|}) \\ &= e^\star \end{aligned}$$

and $d = \tilde{d}_{\mathrm{co}}(u, \pi, x) \geqslant \tilde{d}(\pi_{2\cdots|\pi|}) = d^\star$. It follows that $\langle e^\star, d^\star \rangle \trianglerighteq \langle e, d \rangle$.

Case 3 ($e(u) - e(\pi_1) < x < e(u)$):

Again, let $\langle e^\star, d^\star \rangle = \mathrm{pair}_{\mathrm{co}}(u, \pi, e(u))$. Now,

$$\begin{aligned} e &= \tilde{e}_{\mathrm{co}}(u, \pi, x) \\ &= e_{\mathrm{co}}(u, \pi_1, x) + \tilde{e}(\pi_{2\cdots|\pi|}) \\ &= e_{\mathrm{co}}(u, \pi_1, e(u)) - e(u) + x + \tilde{e}(\pi_{2\cdots|\pi|}) \\ &= \tilde{e}_{\mathrm{co}}(u, \pi, e(u)) - e(u) + x \\ &= e^\star - e(u) + x. \end{aligned}$$

Similarly,

$$\begin{aligned} d &= \tilde{d}_{\mathrm{co}}(u, \pi, x) \\ &\geqslant \tilde{d}_{\mathrm{co}}(u, \pi, e(u)) - e(u) + x \\ &= d^\star - e(u) + x. \end{aligned}$$

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

□

We can now prove Lemma 3.

Proof of Lemma 3 From Eq. (12) and (14) we know that $\mathrm{tdbf}^\star_{\mu_j \to \mu_i}(\tau, \ell)$ is defined by a subset of the set of demand pairs defining $\mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell)$. It follows directly that $\mathrm{tdbf}^\star_{\mu_j \to \mu_i}(\tau, \ell) \leqslant \mathrm{tdbf}^{\mathrm{ub}}_{\mu_j \to \mu_i}(\tau, \ell)$, and the $\Longleftarrow$ direction of the lemma holds.

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

$$\sum_{\tau \in T} e_\tau > \ell_1 \quad \text{and} \quad \max_{\tau \in T}(d_\tau) \leqslant \ell_1. \tag{17}$$

From Lemma 5 we know that for each of the demand pairs $\langle e_\tau, d_\tau \rangle$ there exists some demand pair $\langle e^\star_\tau, d^\star_\tau \rangle \in \mathrm{nco}_{\mu_j \to \mu_i}(\tau) \bigcup \mathrm{co}^\star_{\mu_j \to \mu_i}(\tau)$ such that $\langle e^\star_\tau, d^\star_\tau \rangle \trianglerighteq \langle e_\tau, d_\tau \rangle$. By Definition 5 we have

$$\sum_{\tau \in T} e^\star_\tau - \sum_{\tau \in T} e_\tau \;\geqslant\; \max_{\tau \in T}(d^\star_\tau) - \max_{\tau \in T}(d_\tau). \tag{18}$$

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

$$\sum_{\tau \in T} e_\tau^\star > \max_{\tau \in T}(d_\tau^\star). \tag{19}$$

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

$$\sum_{\tau \in T} \mathrm{tdbf}^\star_{\mu_j \to \mu_i}(\tau, \ell_2) \geqslant \sum_{\tau \in T} e_\tau^\star > \ell_2. \qquad \square$$

Appendix D: Table of Notations Used for the Analysis

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

References

Baruah, S. (2012). Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *SIES*, pages 31–38.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2012). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS*, pages 145–154.

Baruah, S., Bonifaci, V., D'Angelo, G., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2011a). Mixed-criticality scheduling of sporadic task systems. In *ESA*, pages 555–566.

Baruah, S., Burns, A., and Davis, R. (2011b). Response-time analysis for mixed criticality systems. In *RTSS*, pages 34 –43.

Baruah, S., Chen, D., Gorinsky, S., and Mok, A. (1999). Generalized multiframe tasks. *Real-Time Systems*, 17:5–22.

Baruah, K., S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L. (2011c). Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*.

Burns, A. (2014). System mode changes - general and criticality-based. In *WMC*.

Burns, A. and Baruah, S. (2013). Towards a more practical model for mixed criticality systems. In *WMC*.

Burns, A. and Davis, R. (2015). Mixed criticality systems: A review (fifth edition). *Available online at* http://www-users.cs.york.ac.uk/~burns/review.pdf.

Easwaran, A. (2013). Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *RTSS*, pages 78–87.

Ekberg, P. and Yi, W. (2012). Bounding and shaping the demand of mixed-criticality sporadic tasks. In *ECRTS*, pages 135–144.

Ekberg, P. and Yi, W. (2014). Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems*, 50(1):48–86.

Guan, N., Ekberg, P., Stigge, M., and Yi, W. (2011). Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 13–23.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231 – 274.

Huang, P., Giannopoulou, G., Stoimenov, N., and Thiele, L. (2014). Service adaptions for mixed-criticality systems. In *ASP-DAC*, pages 125–130.

Jan, M., Zaourar, L., and Pitel, M. (2013). Maximizing the execution rate of low-criticality tasks in mixed criticality system. pages 43–48.

Li, H. and Baruah, S. (2010). An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 183 –192.

Santy, F., Raravi, G., Nelissen, G., Nelis, V., Kumar, P., Goossens, J., and Tovar, E. (2013). Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In *RTNS*, pages 183–192.

Stigge, M., Ekberg, P., Guan, N., and Yi, W. (2011). The digraph real-time task model. In *RTAS*, pages 71–80.

Stigge, M. and Yi, W. (2013). Combinatorial abstraction refinement for feasibility analysis. In *RTSS*, pages 340–349.

Su, H. and Zhu, D. (2013). An elastic mixed-criticality task model and its scheduling algorithm. In *DATE*, pages 147–152.

Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, pages 239–243.

Zhang, T., Guan, N., Deng, Q., and Yi, W. (2014). On the analysis of EDF-VD scheduled mixed-criticality real-time systems. In *SIES*, pages 179–188.