Budget allocations for hierarchical fixed-priority scheduling of sporadic tasks with deferred preemptions upon EDP resources

[Updated version]*

Martijn M.H.P. van den Heuvel, Reinder J. Bril and Johan J. Lukkien
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven (TU/e)
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

ABSTRACT

In this paper we revisit the admission of applications upon a processor share modeled by the explicit-deadline periodic (EDP) resource-supply model. In particular, we consider applications that represent a fixed-priority sporadic task system. Existing works heavily build on the analysis of a hierarchy of preemptive task schedulers. We instead consider the feasibility of such tasks and applications for a hierarchy of deferred-preemptive schedulers, so that we can efficiently deal with the scenario where tasks and applications execute their work in non-preemptive chunks. Our model therefore gives better control over preemptions of tasks of different applications.

We present exact analysis for deferred-preemptive scheduling of tasks on EDP resources. In addition, we propose algorithms for dimensioning an application's budget tightly.

1. INTRODUCTION

Hierarchical scheduling frameworks (HSFs) have been introduced by Deng and Liu [10] to support preemptive processor sharing among applications under different scheduling policies. The HSF provides means for composing a complex system from well-defined parts called *virtual platforms*. Each virtual platform hosts an application with its own scheduler for scheduling internal workloads (tasks) and system resources are allocated by a global scheduler to the virtual platforms. An HSF essentially provides a mechanism for timing-predictable *composition* of coarse-grained virtual platforms. These virtual platforms can be independently developed, analyzed and tested. Temporal isolation between applications is provided through budgets which are allocated to the corresponding virtual platform.

This work has been presented in the 6th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2013) and is reformatted for appearance in a special issue of the ACM SIGBED Review.

Looking at existing industrial real-time systems, fixed-priority scheduling is the de facto standard for task scheduling. We therefore focus on HSFs with fixed task priorities within a virtual platform. Having such support will simplify migration to and composition of legacy applications into the HSF. Our current research efforts are directed towards the conception and realization of a two-level HSF that is based on

1. fixed-priority scheduling for both *global scheduling* of budgets allocated to virtual platforms and *local scheduling* of tasks within an application,
2. the explicit-deadline periodic (EDP) resource-supply model [12] for computing the sizes of the allocated budgets to applications running on a virtual platform, and
3. the ability to execute tasks for a contiguous duration non-preemptively, both locally and globally.

The latter allows tasks from different applications to share resources requiring mutually exclusive access. An HSF without such support is not realistic, since the tasks of several applications may for example use operating system services, memory mapped devices and shared communication devices. However, preemptive scheduling requires a resource access protocol, such as the stack resource policy (SRP) [1], and an extension of its analysis. As an alternative, fixed-priority scheduling with deferred preemptions of tasks (FPDS) [20] (also known as *co-operative scheduling* [7]) has been proposed as a viable alternative between the extremes of fully preemptive scheduling and non-preemptive scheduling. With FPDS, each task is assumed to consist of a sequence of non-preemptive sub-jobs. When a task is executing, it can be preempted by tasks with a higher priority only between consecutive sub-jobs, i.e., at so-called *preemption points*. Hence, there is no need for a resource access protocol as long as each contiguous mutual-exclusive region is contained in a non-preemptive sub-job.

Compared to both fully preemptive scheduling and non-preemptive scheduling of tasks with fixed priorities, FPDS may significantly improve the feasibility of a task set [20, 4] and, thus, the resource requirements of the constituting application. Moreover, compared to preemptive scheduling, FPDS (*i*) may reduce memory requirements [13] and (*ii*) may reduce the cost of arbitrary preemptions [7, 20, 23, 3, 5], for example, due to cache-related preemption delays and pipeline flushes. However, these advantages of FPDS may no

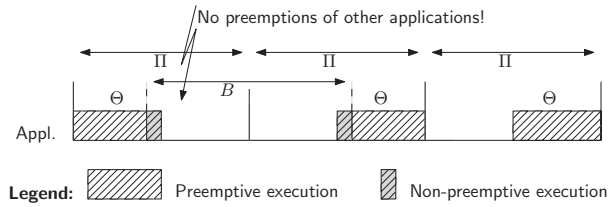

Figure 1: Given an application with a processor budget of Θ time units every period Π. If this application depletes its budget while it executes non-preemptively, excessive blocking durations (B) can occur for other applications.

longer hold when an application is executed upon a virtual platform that can be preempted by other virtual platforms at arbitrary moments in time.

We therefore propose an extension of the EDP resource-supply model [12] in order to be able to analyze hierarchical FPDS of tasks as efficiently as we can analyze it on a dedicated platform (see [20, 4]). When an application is suspended during its non-preemptive execution due to the exhaustion of the corresponding budget, excessive blocking periods can occur (see Figure 1) which may hamper the timeliness of other applications [14]. For this purpose, just like (amongst others) Deng and Liu [10] did, we consider the *overrun* mechanism of Ghazalie and Baker [14] to prevent the depletion of a budget during global non-preemptive execution of a task. An overrun temporarily increases the budget with a statically determined amount for the duration of a non-preemptive sub-job. Global non-preemptive execution of tasks does therefore not only defer preemptions of other tasks of the same application, but it may also defer preemptions of tasks of other applications, although in a controlled manner.

Contributions

In this paper, we first present a new *compositional scheduling model* for tasks and applications that execute their workload in non-preemptive chunks (sub-jobs). We base our model on the following bricks: (*i*) the sporadic task model, (*ii*) fixed task priorities, (*iii*) the EDP resource-supply model and (*iv*) an overrun mechanism for completing non-preemptive sub-jobs of tasks upon budget expiry.

Secondly, we present exact FPDS analysis for the tasks of an application that are allocated an EDP resource.

Finally, we derive an algorithm to compute tight budgets for these applications consisting of limited-preemptive tasks. The following application parameters are our inputs:

- sporadic-task characteristics, as well as the lengths of non-preemptive sub-jobs and relative task priorities.
- a period (Π) and a deadline (Δ) for the allocation of budget (Θ) and overrun budget (χ) which are to be computed.

The overrun budget χ is derived from just task characteristics. Our algorithm computes the smallest possible Θ. Thus, assuming an EDP resource supply of Θ time units every period Π prior to deadline Δ and assuming a bounded overrun of at most χ time units in each period Π, the tasks are guaranteed to meet their deadlines. Further, a periodic resource supply of $\Theta - \varepsilon$, for any infinitesimally small $\varepsilon > 0$, may fail task deadlines.

2. REAL-TIME SCHEDULING MODEL

This section presents a basic, continuous scheduling model, i.e., we assume time to be taken from the real domain ($\mathbb{R}$), similar to, e.g., [20, 4]. We assume a single processor which may be shared by different applications. We first describe the model of FPDS for the scheduling of one application. Secondly, we describe the EDP resource-supply model, and its augmentation, that we use to allocate a virtual platform to each application.

2.1 Application model

An application comprises a set $\mathcal{T}$ of n independent, sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$, with unique priorities $\pi_1, \pi_2, \ldots, \pi_n$. At any moment in time, the processor is used to execute the highest priority task that has work pending. For notational convenience, we assume that (*i*) tasks are given in order of decreasing priorities, i.e., τ_1 has highest priority and τ_n has lowest priority, and (*ii*) a higher priority is represented by a higher value, i.e., $\pi_1 > \pi_2 > \ldots > \pi_n$.

Each task τ_i is characterized by a *minimal inter-arrival time* $T_i \in \mathbb{R}^+$, a *worst-case computation time* $C_i \in \mathbb{R}^+$, and a (*relative*) *deadline* $D_i \in \mathbb{R}^+$. The deadline D_i may be smaller than, equal to, or larger than the period T_i. A release of a task is called a *job*. A job can arrive at an arbitrary time and two subsequent job arrivals are separated by at least T_i time units.

We also adopt standard basic assumptions [17], i.e., tasks do not suspend themselves, a job of a task does not start before its previous job is completed, and the overhead of context switching and task scheduling is ignored.

In FPDS, each task τ_i consists of a pre-defined sequence of m_i non-preemptive sub-jobs, where $m_i \geq 1$. The a-th sub-job of τ_i is denoted by $\tau_{i,a}$ and characterized by a worst-case computation time $C_{i,a} \in \mathbb{R}^+$, where $C_i = \sum_{1 \leq a \leq m_i} C_{i,a}$. Since sub-jobs are non-preemptive, a task can only be preempted between consecutive sub-jobs, i.e., at so-called *preemption points*. FPDS has non-preemptive scheduling as a special case when $(\forall i : 1 \leq i \leq n : m_i = 1)$ and it has fully preemptive scheduling as a special case when the sizes of $C_{i,a}$ are infinitesimally small.

2.2 Virtual platform

A two-level hierarchical scheduling framework (HSF) – as proposed by Deng and Liu [10] – implements a set of N applications $A_1, \ldots, A_N$ on a single processor. These applications abstract the workload generated by their tasks deeper in the hierarchy that implement the real work to be executed. A *global* (top-level) scheduler arbitrates access of an application to the processor. A *local* scheduler determines which task of the selected application executes on the processor. Admitting an application in the HSF should preserve its tasks' deadlines.

The parameters that define a worst-case processor supply to an application, contemplating that timing constraints of the task set are satisfied, together form a so-called *real-time interface*. This interface serves as a *resource supply contract*, which abstracts the timing constraints of the tasks of an application into a single real-time constraint. We model the processor-supply contract by means of an (augmented) *explicit-deadline periodic* (EDP) resource-supply model [12].

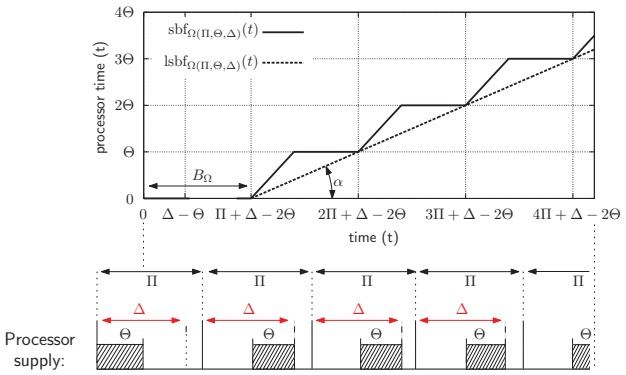

Figure 2: The worst-case EDP supply to an application, $\mathtt{sbf}_\Omega(t)$, in an arbitrary time interval of length t, and its corresponding linear approximation, $\mathtt{lsbf}_\Omega(t)$, according to the EDP model $\Omega(\Pi,\Theta,\Delta)$.

An EDP resource $\Omega(\Pi,\Theta,\Delta,\chi)$ supplies Θ time units of processor time every period of Π time units and Θ is provisioned no later than Δ time units from the start of a period Π (where $\Theta \leq \Delta \leq \Pi - \chi$). An application that receives its processor resources through a virtual platform according to an EDP resource supply interferes with other applications in the system as if it was as a single deadline-constrained periodic task. Further, we augment the default EDP resource-supply model of Easwaran et al. [12] with a parameter χ in order to specify that, in a period Π, an application may request (even more than once) to execute on the processor for at most χ contiguous time units non-preemptively.

The *supply bound function*, $\mathtt{sbf}_{\Omega(\Pi,\Theta,\Delta,\chi)}(t)$, returns the least amount of processor resources supplied to an application in any sliding interval of length t, i.e.,

$$\mathtt{sbf}_\Omega(t) = \max \left\{ \begin{array}{l} 0, \\ \left(h_{(\Omega,t)} - 1\right)\Theta, \\ t - \left(h_{(\Omega,t)} + 1\right)(\Pi - \Theta) + (\Pi - \Delta) \end{array} \right\}, \quad (1)$$

where $h_{(\Omega,t)} = \left\lceil \frac{t-(\Delta-\Theta)}{\Pi} \right\rceil$.

A key property of the $\mathtt{sbf}_\Omega(t)$ is that it is unaware of the details of the application. Hence, it does not take the worst-case contiguous execution demand of an application into account, as described by χ. The $\mathtt{sbf}_\Omega(t)$ is illustrated in Figure 2. The first clause of the max-term prevents that the supply bound becomes negative. The second and the third clause represent alternative views on the available processor time to the application:

1. **application's view:** In an arbitrary interval of length t, the application receives *"k times* Θ" time units. The application starts requesting for processor time when Θ has just been delivered, so that it has to wait for the longest possible duration until processor time is supplied.
2. **platform's view:** In an interval of length t, the application receives *t minus the unavailable processor time*. The worst-case unavailable processor time is characterized as follows: firstly, the delivery of processor resources is blocked for a duration of $B_\Omega = \Pi + \Delta - 2\Theta$ time units and $\Pi - \Theta$ time units are made unavailable periodically.

When an application has the entire platform at its disposal, then we simply ignore the constraints imposed by an augmented EDP model and the processor supply then becomes $\mathtt{sbf}_\Omega(t) = t$.

3. FPDS ANALYSIS ON AN ENTIRE PROCESSOR

In this section, we briefly recapitulate the worst-case schedulability analysis for FPDS as described in [20, 4], i.e., for a continuous scheduling model. We start this section with basic terminology and definitions. Next, we describe the worst-case blocking of tasks under FPDS and recapitulate the notion of ε-critical instant, on which the analysis is based. We conclude this section with an algorithm to determine whether or not a set of sporadic tasks meets its deadlines under FPDS.

3.1 Blocking between the tasks of an application

For a task τ_i, the worst-case blocking, B_i, is given by the largest computation time of a sub-job of a task τ_ℓ with a priority lower than π_i.

$$B_i = \max\left(0, \max_{\ell:\pi_\ell<\pi_i} \max\left\{ C_{\ell,a} \;\middle|\; 1 \leq a \leq m_\ell \right\}\right). \quad (2)$$

The outermost max in (2) is used to define B_i in situations where there exists no lower priority task with a sub-job that prevents τ_i to preempt, e.g., for the lowest priority task τ_n.

Also, a sub-job of a low priority task τ_ℓ must start with its execution of B_i prior to the release of the higher priority job of task τ_i. Therefore, if this blocking time B_i is positive, it is a *supremum* and not a maximum.

3.2 Critical instant and level-i active period

A *critical instant* of a task is defined to be a (hypothetical) instant that leads to the worst-case response time for that task [17]. The maximum response time of a task τ_i can be found when (*i*) τ_i has a simultaneous release with all higher priority tasks and (*ii*) the longest sub-job of the lower priority tasks with computation time B_i starts an infinitesimally small time ε before that simultaneous release. However, the maximum response time is not necessarily the first job of task τ_i after a critical instant, but can occur in later jobs contained within a so-called *level-i active period* [20][1].

To define the level-i active period, we use the notion of pending load. The pending load $P_i(t)$ is the amount of processing at time t that still needs to be performed for the jobs with a priority higher than or equal to task τ_i that are released before time t. A level-i active period is an interval $[t_s, t_e)$, such that $P_i(t) > 0$ for all $t \in (t_s, t_e)$ and $P_i(t_s) = P_i(t_e) = 0$.

The worst-case length $t_e - t_s$ of a level-i active period is denoted by WL_i. This length WL_i is computed through a recurrence [20]

$$x^{(\ell+1)} = B_i + \sum_{j:\pi_j \geq \pi_i} \left\lceil \frac{x^{(\ell)}}{T_j} \right\rceil C_j, \quad (3)$$

where $x^{(0)} = B_i + C_i$. The recursion stops when $x^{(\ell+1)} = x^{(\ell)}$, yielding the worst-case length WL_i of a level-i active period. The proced-

[1]The notion of level-i active period supersedes the notion of level-i busy period [15]; see [20].

ure is guaranteed to terminate when the utilization $U^{\mathcal{T}}$ of the task set $\mathcal{T}$ is less than one, i.e., $U^{\mathcal{T}} = \sum_{1 \le i \le n} \frac{C_i}{T_i} < 1$.

The worst-case number of jobs of task τ_i released in a level-i active period, denoted by wl_i, is given by

$$wl_i = \left\lceil \frac{WL_i}{T_i} \right\rceil. \tag{4}$$

The worst-case response time of task τ_i is then given by the maximum of the response times of the jobs $k \in [0, wl_i)$.

3.3 Determining schedulability of an application

We now recapitulate the schedulability test for FPDS based on discontinuous points of cumulative execution-request functions [4]. The worst-case cumulative execution request $W(\pi,t)$ in an interval $[a,b)$ of length t of all tasks with a priority higher than π is given by

$$W(\pi,t) \stackrel{\text{def}}{=} \sum_{h:\pi_h > \pi} \left\lceil \frac{t}{T_h} \right\rceil C_h. \tag{5}$$

The modified execution request $W^*(\pi,t)$ in a closed interval of length t is given by

$$W^*(\pi,t) \stackrel{\text{def}}{=} \sum_{h:\pi_h > \pi} \left(\left\lfloor \frac{t}{T_h} \right\rfloor + 1 \right) C_h. \tag{6}$$

We use $W(\pi,t)$ and $W^*(\pi,t)$ in a function $\psi_{i,k}(x)$ that defines the cumulative execution request of a job k of task τ_i (until its last sub-job) and all higher priority jobs, (excluding the blocking term), i.e.,

$$\psi_{i,k}(x) = (k+1) \times C_i - C_{i,m_i} + \begin{cases} W(\pi_i,x) & \text{for } B_i > 0 \\ W^*(\pi_i,x) & \text{for } B_i = 0 \end{cases}. \tag{7}$$

We write $\psi_{i,k}\Big|_{(B_i>0)}(t)$ and $\psi_{i,k}\Big|_{(B_i=0)}(t)$ in order to refer to the distinctive cases in (7) for the $B_i > 0$ and $B_i = 0$, respectively.

The time points to be inspected, representing the discontinuous points of $\psi_{i,k}(x)$, can be represented as

$$\Pi_{i,k} = (k \times T_i,\ k \times T_i + D_i - C_{i,m_i}] \bigcap \{h \times T_j \mid \forall h \in \mathbb{N},\ j \le i\} \bigcup \{k \times T_i + D_i - C_{i,m_i}\}. \tag{8}$$

$\Pi_{i,k}$ is a non-empty and a finite set of time points: it contains at least $\left(k \times T_i + D_i - C_{i,m_i}\right)$. In special cases, e.g., $C_i = D_i$, this may include time 0 for job $k = 0$. Note that the inspected time points, presented by Bertogna et al. [4], do not necessarily coincide with the exact worst-case start of the final sub-job τ_{i,m_i} of job k of task τ_i as they are computed by Bril et al. [20].

We are now ready to present an exact schedulability test for FPDS using execution-request curves.

THEOREM 1 (THEOREM 1 IN [4]). *A task set is schedulable under FPDS, if and only if* $\forall i : 1 \le i \le n$:

$$\forall k \in [0, wl_i) :: \left(\exists t \in \Pi_{i,k} :: B_i + \psi_{i,k}(t) \le t\right) \tag{9}$$

Algorithm 1 jobTolerance($\mathcal{T}$, i, k, C_{i,m_i})

```
1: β_{i,k} ← max_{t∈Π_{i,k}} { t − ψ_{i,k}|_{(B_i>0)}(t) };
2: if β_{i,k} = 0 then
3:     t* = k × T_i + D_i − C_{i,m_i};
4:     β_{i,k} ← t* − ψ_{i,k}|_{(B_i=0)}(t*);
5: end if
6: return β_{i,k};
```

Algorithm 2 computeBlockingTolerance($\mathcal{T}$, i, C_{i,m_i})

```
1: {Find the blocking tolerance for the first job:}
2: β_{i,0} ← jobTolerance(T, i, 0, C_{i,m_i});
3: if β_{i,0} < 0 then return β_{i,0}
4: β_i ← β_{i,0};
5: compute ŵl_i using (15);
6: {Find the minimum β_{i,k} in the level-i active period:}
7: for k ← 1; k < ŵl_i; k ← k + 1 do
8:     β_{i,k} ← jobTolerance(T, i, k, C_{i,m_i});
9:     if β_{i,k} < 0 then return β_{i,k}
10:    β_i ← min(β_i; β_{i,k});
11: end for
12: return β_i;
```

and for every task τ_i *with* $B_i = 0$, *for each job* k *there exists a* $t \in \Pi_{i,k}$ *such that the following condition holds:*

$$\psi_{i,k}\Big|_{(B_i>0)}(t) < t \ \bigvee \ \left(\begin{matrix} t^\star = k \times T_i + D_i - C_{i,m_i} \\ \psi_{i,k}\Big|_{(B_i=0)}(t^\star) \le t^\star \end{matrix} \right). \tag{10}$$

PROOF. See Theorem 1 in [4]. □

We conclude that a nice property of FPDS is that a task may accelerate its own completion, at the cost of blocking others, by finishing its final sub-job, C_{i,m_i}, without any interference.

3.4 Blocking tolerance of tasks

Bertogna et al. [4] have proposed an efficient algorithm to implement their schedulability test (see Theorem 1) based on so-called blocking tolerances [18] of tasks. The blocking tolerance β_i of a task τ_i, as introduced by Lortz and Shin [18], is the maximum amount of blocking that a lower priority task may induce to τ_i without hampering the feasibility of task τ_i. The blocking tolerance for a task τ_i is defined by the minimum blocking tolerance of all its jobs in a level-i active period:

$$\beta_i = \min_{0 \le k < wl_i} \left\{ \beta_{i,k} \right\}, \tag{11}$$

where $\beta_{i,k}$ is the blocking tolerance of job k of τ_i.

Based on Theorem 1, the schedulability of job k of τ_i can be checked for $B_i > 0$ by

$$\exists t \in \Pi_{i,k} \left(B_i \le t - \psi_{i,k}(t) \right). \tag{12}$$

Equation (12) can be rewritten into the blocking tolerance $\beta_{i,k}$ of job k of τ_i, as follows:

$$\beta_{i,k} = \max_{t \in \Pi_{i,k}} \left\{ t - \psi_{i,k}(t) \right\}. \tag{13}$$

The definition for $\beta_{i,k}$ is only correct for a strictly positive blocking tolerance (i.e. $B_i > 0$). In case that $\beta_{i,k} < 0$, we know that job k deems the task unschedulable. When $\beta_{i,k} = 0$, then we need to verify whether or not $\beta_{i,k}$ is really equal to 0 using the rules for $B_i = 0$ in Theorem 1. Algorithm 1 presents the algorithm derived by Bertogna et al. [4] for determining the blocking tolerance of job k of task τ_i.

After computing the blocking tolerance for the first job, $\beta_{i,0}$, we have an upper bound for the maximum amount of blocking that task τ_i may suffer, see (11). We can therefore use the value $\beta_{i,0}$ to compute an upper bound on the worst-case length, see (3). That is, in (3) we replace B_i by a larger value $\beta_{i,0}$. An upper bound on worst-case length $\widehat{WL_i}$ is then given by the smallest $x \in \mathbb{R}^+$ that satisfies the following recursive equation

$$x^{(\ell+1)} = \beta_{i,0} + \sum_{j:\pi_j \geq \pi_i} \left\lceil \frac{x^{(\ell)}}{T_j} \right\rceil C_j, \tag{14}$$

where $x^{(0)} = \beta_{i,0} + C_i$. The recursion stops when $x^{(\ell+1)} = x^{(\ell)}$, yielding a upper bound $\widehat{WL_i}$ of a level-i active period.

A worst-case bound on the number of jobs of task τ_i released in a level-i active period, denoted by $\widehat{wl_i}$, is then given by

$$\widehat{wl_i} = \left\lceil \frac{\widehat{WL_i}}{T_i} \right\rceil. \tag{15}$$

Having obtained this bounded number of $\widehat{wl_i}$ jobs to be analyzed, Algorithm 2 simply computes the blocking tolerance β_i by taking the minimum over all jobs to be considered for task τ_i, i.e., according to (11).

Finally, the main result derived by Bertogna et al. [4] is that a set of n sporadic tasks, with given task characteristics $(T_i, D_i, C_{i,1}, \ldots C_{i,m_i})$, can be scheduled by FPDS, if and only if

$$\forall i \ : 1 \leq i \leq n \ : 0 \leq B_i \leq \beta_i, \tag{16}$$

where B_i can be computed as in (2) and β_i can be computed using Algorithm 2.

We will later re-use Algorithm 2 in order to decide whether or not a set of tasks can also be scheduled by FPDS upon a virtual platform. This is more complicated than the analysis of an application upon an entire processor, because tasks may not only experience blocking from lower-priority tasks of the same application, but also from tasks in other applications and from the unavailability of platform resources, B_Ω.

4. COMPOSITIONAL ANALYSIS FOR HIERARCHICAL FPDS

This section presents compositional analysis for hierarchical FPDS of tasks. Firstly, we consider the global schedulability test which implements the admission control of applications based on their resource requirements specified by augmented EDP interfaces. During the composition of applications, we take into account the blocking by tasks located in other applications. Secondly, we consider the local schedulability analysis of the tasks of an application upon an EDP resource. During the application analysis, we take into account the blocking due to the unavailability of the platform resources.

4.1 Composition of virtual platforms

We consider an *overrun* mechanism [9, 2, 21] to prevent depletion of a budget during global non-preemptive execution of a task by temporarily increasing the budget with a statically determined amount for the duration of that non-preemptive sub-job. To distinguish this additional amount of budget from a *normal budget* (Θ), we will use the term *overrun budget*. Since an application with an interface $\Omega(\Pi, \Theta, \Delta, \chi)$ may request at most χ time units of contiguous non-preemptive execution, its overrun budget is of the same size as χ. Note that a special property of the overrun budget is that it is provisioned entirely non-preemptively. We will exploit this property in our analysis.

Given an EDP interface $\Omega(\Pi, \Theta, \Delta, \chi)$, we therefore derive a sporadic task τ_i with the following timing characteristics:

- $T_i \leftarrow \Pi$;
- $C_i \leftarrow (\Theta + \chi)$;
- $D_i \leftarrow (\Delta + \chi)$;
- $C_{i,m_i} \leftarrow \chi$.

In line with the FPDS task model, we re-use the FPDS analysis of [20, 4] at the scheduling level of virtual platforms allocated to applications. The conventional EDP [12] parameter Δ serves as a deadline for the supply of just the normal budget Θ. Beyond Δ, only an overrun budget χ may be supplied and, if it is supplied, it is supplied non-preemptively. Hence, Δ is just an *intermediate deadline* [7] for budget Θ and the total worst-case supply $(\Theta + \chi)$ is constrained by a relative deadline $(\Delta + \chi) \leq \Pi$.

Assume a set of applications $A_1 \ldots A_N$ with corresponding interfaces $\Omega_1 \ldots \Omega_N$ and a fixed priority assignment to the applications[2]. Using the above characteristics of an application in terms of the classic sporadic task model, we can now apply the schedulability analysis presented in Section 3.

In order to apply the analysis in (16), we must compute the blocking B_i experienced by an application A_i, caused by the execution of a non-preemptive sub-job of a task in a lower priority application A_ℓ. For ease of presentation, we assume that applications are ordered by descending priorities, i.e., A_1 has the highest priority and A_N has the lowest priority. The blocking term B_i can then be instantiated from (2) as follows:

$$B_i = \max\{\chi_\ell \mid \ell > i\}. \tag{17}$$

We now have the timing information which allows us to test the condition in (16) for a given set of applications.

4.2 Analyzing an application upon a virtual platform

We now analyze the tasks of an application that have to run on a virtual platform. In order to be able to reuse the existing analysis of FPDS for application mappings on EDP resources, we must take into account the scheduling delays imposed by the unavailability of the platform resources.

Looking at Figure 2, there are two types of scheduling delays that may come with an EDP resource:

[2]Davis and Bertogna [8] have recently proposed an algorithm to determine the priorities of a set of tasks optimally under FPDS. Since applications (are forced to) behave like tasks, their algorithm can be applied to our scheduling model at the top-level as well.

1. a delay, B_Ω, of the start of job executions;

2. a periodic delay, $\Pi - \Theta$, due to budget depletion.

Firstly, as we have discussed before, the scheduling delay of B_Ω time units is an additional blocking term for the tasks of an application. This means that a necessary condition for fitting an application on an EDP resource Ω is that the total blocking, either by the virtual platform (B_Ω) or by local lower priority tasks (B_i) inside the application, must not exceed the blocking tolerance of a task. Formally formulated:

$$\forall i : 1 \leq i \leq n : 0 \leq B_\Omega + B_i \leq \beta_i, \tag{18}$$

where B_i is defined in (2) and β_i is defined in (11).

Secondly, budget depletion causes a preemption of the running application at an arbitrary position in the execution of its tasks. This property of the EDP model fits a fully preemptive scheduling model (as it is considered by Easwaran et al. [12]). Under FPDS of tasks, however, this property is undesirable at run time, because the sub-jobs of tasks are meant to be non-preemptive. We therefore allow budget overruns (for a duration of at most χ time units) in order to complete the execution of a non-preemptive sub-job, so that arbitrary preemptions of tasks are prevented. This application behavior can only be observed during online execution, however. Both the resource-supply model of a virtual platform (see Section 2.2) and the task-level analysis of an application (see Section 3) abstract from this behavior. Inherited from the FPDS analysis in [20, 4] (see Section 3), our analysis assumes a fully preemptive scheduling model for tasks until they start executing their final sub-job. From then onwards, this job must finish non-preemptively.

Since FPDS forbids further delays of the execution of a task's last sub-job, it is important to note that budget overruns are instrumental for the reuse of the existing analysis of FPDS on EDP resources. As we can see in (7), the requested resources of a task τ_i are only measured until the last preemption point, i.e., preemptions are accounted until the start of the final sub-job (of length C_{i,m_i}). From this preemption point onwards, it is assumed that this task will complete within C_{i,m_i} time units, because sub-jobs cannot be preempted. Hence, our implementation of a virtual platform must give the same guarantees to this task, e.g., through an overrun mechanism (as we propose in this paper). This leads us to the following analysis for an application scheduled under hierarchical FPDS on an EDP resource.

THEOREM 2. *A set of sporadic tasks is schedulable under FPDS upon an EDP resource $\Omega(\Pi,\Theta,\Delta,\chi)$ with an overrun of at most χ time units every period Π, if $\forall i : 1 \leq i \leq n : \forall k \in [0, wl_i) ::$*

$$\left(\exists t \in \Pi_{i,k} :: B_i + \psi_{i,k}(t) \leq \mathtt{sbf}_\Omega(t)\right) \tag{19}$$

and

$$\left(\forall i, j : 1 \leq i \leq n \bigwedge 1 \leq j \leq m_i : C_{i,j} \leq \chi\right) \tag{20}$$

and for every task τ_i with $B_i = 0$, for each job k there exists a $t \in \Pi_{i,k}$ such that the following condition holds:

$$\psi_{i,k}\Big|_{(B_i>0)}(t) < \mathtt{sbf}_\Omega(t) \bigvee \left(\begin{array}{l} t^\star = k \times T_i + D_i - C_{i,m_i} \\ \psi_{i,k}\Big|_{(B_i=0)}(t^\star) \leq \mathtt{sbf}_\Omega(t^\star) \end{array}\right). \tag{21}$$

PROOF. Similar to Theorem 1 in [4]. □

Under FPDS it is assumed that if the last sub-job of a task τ_i starts, then it will also complete within C_{i,m_i} time units. Hence, if non-preemptive execution of a sub-job is initiated and the budget Θ depletes, then χ must be supplied. For this purpose, we introduced the constraint in (20) in order to make sure that the allocated overrun budget χ is sufficiently large to meet the FPDS assumption on the progress of the task's execution.

5. DESIGNING A VIRTUAL PLATFORM FOR AN APPLICATION

In this section we present algorithms to compute the smallest possible budgets for a given application. Assume we are given an application consisting of a set of tasks $\mathcal{T}$ with given characteristics $(T_i,\ D_i,\ C_{i,1} \ldots C_{i,m_i},\ \pi_i)$. We select the period parameter Π and the deadline parameter Δ as the design parameters in the application's augmented EDP interface. Given these inputs, the question is then: what are the smallest (normal) budget Θ and the smallest overrun budget χ satisfying the scheduling conditions in Theorem 2?

In this section we answer this question. Let us first define the notion of an exact EDP interface.

DEFINITION 1 (EXACT EDP INTERFACE). *Assume a given application consisting of a task set $\mathcal{T}$ with given timing characteristics $(T_i,\ D_i,\ C_{i,1} \ldots C_{i,m_i},\ \pi_i)$, a given period Π and a given deadline Δ. We call an EDP interface $\Omega(\Pi,\Theta,\Delta,\chi)$ exact (or optimal) for this application, if*

- *the schedulability conditions of Theorem 2 are satisfied with a budget Θ and an overrun budget χ.*
- *those conditions are violated with $\Theta - \varepsilon$ or with $\chi - \xi$, for any infinitesimally small $\varepsilon > 0$ and $\xi > 0$.*

The computation of Θ closely follows the classic analysis of FPDS [20, 4], just like our analysis for hierarchical FPDS in Section 4 does. **Although Theorem 2 is not an exact scheduling test for hierarchical FPDS, the notion of an exact interface refers to the tightest budget we can compute by means of this theorem.** This means that, during the computation of budget Θ, we cannot assume that overrun is provisioned during the execution of tasks unless we encounter the situation where the last sub-job of a task has been started.

5.1 Allocating an overrun budget χ

Using Definition 1, we can straightforwardly allocate an overrun budget χ to an application. The following assignment of χ satisfies the necessary schedulability condition in (20):

$$\chi \leftarrow \max\left\{C_{i,j} \;\middle|\; 1 \leq i \leq n \bigwedge 1 \leq j \leq m_i\right\}. \tag{22}$$

5.2 Allocating a periodic budget Θ

Computing the smallest possible budget Θ for an application is more complicated than deriving overrun budgets. Dewan and Fisher [11] have presented an approximation schemes for computing the budgets Θ under fully preemptive scheduling of sporadic tasks upon an EDP resource (assuming a given period Π and a given deadline Δ). Based on their approximation algorithm for fixed-priority preemptive scheduling of tasks, we derive an exact algorithm for computing an optimal budget for FPDS of tasks.

Algorithm 3 ComputePartialBudget(Π, Δ, $Y(t)$, t)

1: $\ell = \frac{(t-\Delta)+\sqrt{(t-\Delta)^2+4\Pi\cdot Y(t)}}{2\Pi}$
2: $\Theta_1 \leftarrow \frac{Y(t)-t+(\lfloor\ell\rfloor+1)\Pi+\Delta}{\lfloor\ell\rfloor+2}$
3: $\Theta_2 \leftarrow \frac{Y(t)}{\lceil\ell\rceil-1}$
4: **if** $\lceil\ell\rceil \leq \lfloor\ell\rfloor$ **then**
5: $\quad\Theta_2 \leftarrow \frac{Y(t)}{\lceil\ell\rceil}$
6: **end if**
7: $\Theta_t^{\min} \leftarrow \min(\Theta_1,\ \Theta_2)$
8: **return** $\Theta_t^{\min}$

Similar to, e.g., [11, 16], we first derive an algorithm to solve the inequalities in the scheduling conditions in (19) and (21).

LEMMA 3. *Given an application with a cumulative workload represented by variable $Y(t)$ at time t, a period Π and a deadline Δ. The smallest budget Θ, satisfying the inequality $Y(t) \leq \mathtt{sbf}_\Omega(t)$, is given by*

$$\Theta \geq \frac{Y(t)}{k} \ \bigvee\ \Theta \geq \frac{-t+Y(t)+k\Pi+\Delta}{k+1} \tag{23}$$

such that $k \in \mathbb{N}^+$ and

$$\frac{Y(t)}{k} < \frac{-t+Y(t)+k\Pi+\Delta}{k+1} \Leftrightarrow k \geq \lfloor\ell\rfloor+1 \tag{24}$$

and

$$\frac{Y(t)}{k} \geq \frac{-t+Y(t)+k\Pi+\Delta}{k+1} \Leftrightarrow \begin{array}{ll} k \leq \lceil\ell\rceil & \text{if } \lceil\ell\rceil < \lfloor\ell\rfloor+1 \\ k \leq \lceil\ell\rceil-1 & \text{else,} \end{array} \tag{25}$$

where

$$\ell = \frac{t-\Delta+\sqrt{(t-\Delta)^2+4\Pi\cdot Y(t)}}{2\Pi}. \tag{26}$$

PROOF. Variable k reconstructs the value of $h_{(\Omega,t)}$ in (1) by computing the intersection between the two non-zero line segments in coordinate $(t,\ Y(t))$, see (23). The intersection is characterized by the roots of the convex parabola $\Pi k^2 + k(\Pi - t) - Y(t)$. The value of k is determined by observing that $h_{(\Omega,t)} \in \mathbb{N}^+$. The real-number representation of the positive root, ℓ, of the parabola is given in (26); its counter part always has a negative value, because the term in the square root dominates the preceding term. Since the left-hand sides of the bi-implications in (24) and (25) are strictly non-overlapping (mutually exclusive) inequalities, we have to guarantee a strictly smaller value $k \in \mathbb{N}^+$ in (25) than in (24). The case distinction detects whether or not $\lceil\ell\rceil = \lfloor\ell\rfloor + 1$. This concludes the proof. □

From Lemma 3, we can directly derive Algorithm 3 to compute an optimal budget for a given period Π and a given deadline Δ, satisfying the requested resources $Y(t)$ at time t.

With the help of Algorithm 3, we can solve the inequalities in the scheduling conditions in (19) and (21). In line with [20, 4], these scheduling conditions present two non-uniform cases for comparing the cumulative processor requirements of a task against the supplied resources: one with and one without a blocking term. As shown before by condition (18), however, the tasks executing on a virtual platform may not only experience blocking from lower priority tasks of the same application, but they also suffer blocking due to the unavailability of the virtual-platform's resources, B_Ω. On a virtual platform, the blocking tolerance β_i of a task τ_i must therefore be larger than 0, because otherwise the corresponding application cannot run on an EDP resource other than the entire processor. This observation simplifies our algorithm for computing EDP budgets, because we can ignore the non-uniform case in (21) without blocking and we focus on solving the inequality in (19).

Algorithm 4 FPDSMinimumBudget($\mathcal{T}$, Π, Δ)

1: $\Theta^{\text{opt}} \leftarrow 0$;
2: **for** $i \leftarrow 0$; $i < n$; $i \leftarrow i+1$ **do**
3: $\quad$ {Compute β_i using Algorithm 2:}
4: $\quad\beta_i \leftarrow$ computeBlockingTolerance($\mathcal{T}$, i, C_{i,m_i});
5: $\quad$**if** $\beta_i < 0 \bigvee B_i > \beta_i$ **then return** infeasible;
6: $\quad$**if** $\beta_i = 0 \bigvee \Theta^{\text{opt}} \geq \Pi$ **then**
7: $\quad\quad${$\mathcal{T}$ cannot run with less than the entire processor:}
8: $\quad\quad\Theta^{\text{opt}} \leftarrow \max(\Theta^{\text{opt}},\ \Pi)$;
9: $\quad\quad$**continue**;
10: $\quad$**end if**
11: $\quad${**Invariant:** $\beta_i > 0$}
12: $\quad$Determine $\widehat{wl_i}$ using (15); {Reuse it from Algorithm 2}
13: $\quad$**for** $k \leftarrow 0$; $k < \widehat{wl_i}$; $k \leftarrow k+1$ **do**
14: $\quad\quad$**for each** $t \in \Pi_{i,k}$ **do**
15: $\quad\quad\quad${Compute the smallest requirement of this job:}
16: $\quad\quad\quad\Theta_{i,k,t} \leftarrow$ ComputePartialBudget($\Pi, \Delta, B_i + \psi_{i,k}(t), t$);
17: $\quad\quad\quad\Theta_{i,k} \leftarrow \min\left(\Theta_{i,k},\ \Theta_{i,k,t}\right)$;
18: $\quad\quad$**end for**
19: $\quad\quad${Take the largest requirement of all jobs of a task:}
20: $\quad\quad\Theta_i \leftarrow \max\left(\Theta_i,\ \Theta_{i,k}\right)$;
21: $\quad$**end for**
22: $\quad${Take the largest requirement of all tasks:}
23: $\quad\Theta^{\text{opt}} \leftarrow \max(\Theta^{\text{opt}},\ \Theta_i)$;
24: **end for**
25: **return** Θ^{opt};

Algorithm 4 computes the smallest possible budget of an EDP resource that still allows tasks to make their deadlines under hierarchical FPDS. It iterates through all the tasks of an application and it works as follows. First (line 3–5), we compute the blocking tolerance of a task and we check whether or not the considered task satisfies the necessary scheduling condition in (16). If the blocking tolerance of a task is 0, then no further blocking from the platform (B_Ω) can be tolerated, i.e., this task requires the entire processor (see line 6–10). In this situation, we do not need to consider further computations of budget Θ, see the *continue* statements (line 9). Finally, in line 11–23 we compute the smallest budget Θ_i that that allows a task to make its deadline.

In order to compute such a budget Θ_i, we must have an upper bound on the number of jobs of task τ_i to be tested for schedulability. Looking at the scheduling condition in (19), we observe also that the level-i active period, WL_i, depends on the worst-case amount of blocking experienced by task τ_i. Since we aim at minimizing the allocated EDP budget, Θ, to an application, we correspondingly try to maximize the amount of blocking by the platform (B_Ω) to the tasks of that application. The freedom we have is restricted by the blocking tolerance of tasks, see (18). Considering this restriction, we reuse the same conservative upper bound $\widehat{WL_i}$ as Bertogna et al. [4] did, i.e., see Algorithm 2 (line 12) and see (14). We now loop

through these jobs (line 13–21) and for each of them we inspect their set of discontinuous time points (line 14–18). Since we can replace the quantifiers $(\forall, \exists)$ in the scheduling conditions of tasks in (19) by max − min operators, we can apply Algorithm 3 to compute the intersection of the execution-request curves with the resource-supply function, $\mathtt{sbf}_\Omega(t)$, at the inspected time points (line 16).

To be precise, the loop in line 13–21 computes the smallest budget Θ_i for a task τ_i. Using the condition (19) in Theorem 2 and filling in the execution-request of task τ_i in condition (23) of Lemma 3, we obtain the following solution: $\Theta_i \geq$

$$\max_{k\in[0,wl_i)} \min_{t\in\Pi_{i,k}} \left\{ \min\left(\frac{B_i+\psi_{i,k}(t)}{k}, \frac{-t+B_i+\psi_{i,k}(t)+k\Pi+\Delta}{k+1} \right) \right\}. \tag{27}$$

and the value of k in (27) is reconstructed using Algorithm 3.

Finally, in order to meet the deadlines of all tasks of an application on an EDP resource, we allocate the largest budget requirements Θ_i (see line 21). Algorithm 4 therefore returns an optimal budget $\Theta^{\mathtt{opt}}$ (line 25) which defines an augmented EDP interface $\Omega(\Pi, \Delta, \Theta^{\mathtt{opt}}, \chi)$, with χ according to (22), that can be used to dimension the virtual platform allocated to the analyzed application.

5.3 Algorithmic complexity of budget allocations

The complexity of determining the overrun budget χ is linear in the number of tasks and linear in the number of sub-jobs of an application, see (22). The complexity of determining budget Θ is more complicated. Nevertheless, Algorithm 4 has the same complexity as the the scheduling analysis of FPDS in [20, 4]. The operations of Algorithm 3, `ComputePartialBudget(...)`, take just constant time complexity. Hence, the overall complexity of Algorithm 4 is determined by the sequence of computing (*i*) the blocking tolerances β_i (line 3–10) and (*ii*) the optimal budget $\Theta^{\mathtt{opt}}$ (line 11–23). Each of these two parts of Algorithm 4 has the same computational complexity, i.e., determined by the number of jobs and the corresponding number of generated time points in $\Pi_{i,k}$ that need to be tested per job. Hence, Algorithm 4 has a pseudo-polynomial time complexity in the number of tasks of an application.

6. CONCLUSION

In this paper we presented a compositional model to enable fixed-priority scheduling of tasks with deferred preemptions across a hierarchy of schedulers. For this purpose, we defined a virtual platform using the EDP resource-supply model, augmented with an overrun mechanism. In our augmented EDP model, the virtual platform satisfies the classic FPDS model. A major advantage of applying the FPDS model is that it considers also the benefits (i.e, accelerated executions) of the tasks that block other tasks of an arbitrary application. However, most literature for hierarchical scheduling builds on preemptive scheduling analysis, for example, see [10, 9, 2, 22], which takes just the penalties of non-preemptive task executions into account. With our new model, we can re-use the analysis of FPDS, and its strengths, for the admission control of applications through virtual platforms on a shared processor.

We also presented an algorithm to compute the sizes of the processor budgets of a virtual platform. That is, given the task characteristics of an application and given a period and deadline constraint for the allocation of a budget, we compute the smallest periodic budget and the smallest overrun budget such that the tasks of an application meet their deadlines on a virtual platform that complies to the computed parameters. Since many resources (for example, a memory bus) require non-preemptive access, we believe that our scheduling model and its corresponding analysis is a useful step towards a resource-efficient composition of resource-sharing applications.

As a future work, we would like to further optimize the composition process of applications. It would be interesting to investigate whether or not the preemption cost of multiple applications that share the same processor can be alleviated, e.g., by finding appropriate placements of preemption points [5] in the execution of tasks or by using preemption thresholds for preemption points of tasks [6]. In addition, it would be interesting for open environments to alleviate the pseudo-polynomial complexity of our scheduling analysis. A novel approximation scheme for budget allocations for hierarchical FPDS may be inspired by, for example, the approximation scheme for hierarchical preemptive scheduling in [11] or the approximation scheme for FPDS in [19].

Acknowledgements

This work was supported in part by the European Union's ARTEMIS Joint Undertaking for CRYSTAL – Critical System Engineering Acceleration – under grant agreement No. 332830.

References

[1] T. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems Journal*, 3(1): 67-99, March 1991.

[2] M. Behnam, T. Nolte, M. Sjodin, and I. Shin. Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems. *IEEE TII*, 6(1):93 –104, Feb. 2010.

[3] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo. Preemption points placement for sporadic task sets. In *ECRTS*, pages 251–260, July 2010.

[4] M. Bertogna, G. Buttazzo, and G. Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *RTSS*, pages 251–260, Dec. 2011.

[5] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *ECRTS*, pages 217–227, July 2011.

[6] R. J. Bril, M. M. H. P. van den Heuvel, and J. J. Lukkien. Improved feasibility of fixed-priority scheduling with deferred preemptions using preemption thresholds for preemption points. In *RTNS*, pages 255–264, Oct. 2013.

[7] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[8] R. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *RTSS*, pages 39–50, Dec. 2012.

[9] R. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *RTSS*, pages 257–267, 2006.

[10] Z. Deng and J.-S. Liu. Scheduling real-time applications in open environment. In *RTSS*, pages 308–319, Dec. 1997.

[11] F. Dewan and N. Fisher. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In *RTAS*, pages 247–256, April 2010.

[12] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *RTSS*, pages 129–138, Dec. 2007.

[13] P. Gai, G. Lipari, and M. Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *RTSS*, pages 73–83, Dec. 2001.

[14] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems Journal*, 9(1):31–67, 1995.

[15] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, pages 201–209, Dec. 1990.

[16] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, 2005.

[17] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a real-time environment. *Journal of the ACM*, 20(1): 46–61, Jan. 1973.

[18] V. Lortz and K. Shin. Semaphore queue priority assignment for real-time multiprocessor synchronization. *IEEE TSE*, 21(10): 834–844, Oct. 1995.

[19] T. H. C. Nguyen, N. S. Tran, V. H. Le, and P. Richard. Approximation scheme for real-time tasks under fixed-priority scheduling with deferred pre-emption. In *RTNS*, pages 265–274, Oct. 2013.

[20] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems Journal*, 42(1-3): 63-119, Aug. 2009.

[21] R.J. Bril, U. Keskin, M. Behnam, and T. Nolte. Schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks revisited. In *CRTS*, Dec. 2009.

[22] R. Santos, P. Pedreiras, M. Behnam, T. Nolte, and L. Almeida. Multi-level hierarchical scheduling in ethernet switches. In *EMSOFT*, pages 9–14, Oct. 2011.

[23] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed-priority scheduling. In *RTCSA*, pages 351–360, Aug. 2009.