A

Adaptive Workload Management in Mixed-Criticality Systems

BIAO HU, Technische Universität München
KAI HUANG, Sun Yat-Sen University and Technische Universität München
GANG CHEN, Northeastern University, China
LONG CHENG, Technische Universität München
ALOIS KNOLL, Technische Universität München

Due to the efficient resource usage of integrating tasks with different criticality onto a shared platform, the integration with mixed-criticality tasks is becoming an increasingly important trend in the design of real-time systems. One challenge in such a mixed-criticality system is to maximize the service for low-critical tasks, while meeting the timing constraints of high-critical tasks. In this article, we investigate how to adaptively manage the low-critical workload during the runtime to meet both goals, that is, providing the service for low-critical tasks as much as possible and guaranteeing the hard real-time requirements for high-critical tasks. Unlike the previous methods that enforce an offline bound towards the low-critical workload, runtime adaption approaches are proposed where the incoming workload of low-critical tasks is adaptively regulated by considering the actual demand of high-critical tasks. This actual demand of the high-critical tasks in turn is adaptively updated using their historical arrival information. Based on this adaption scheme, two scheduling policies, namely the priority-adjustment policy and the workload-shaping policy, are proposed to do the workload management. In order to reduce the online management overheads, a lightweight scheme with $O(n \cdot log(n))$ complexity is developed. Extensive simulation results are presented to demonstrate the effectiveness of our proposed workload management approaches.

Categories and Subject Descriptors: D.4.1 [**Process Management**]: Real-Time Task Priority Assignment

General Terms: Workload Management, Online

Additional Key Words and Phrases: Workload management, mixed-criticality systems, real-time event streams, real-time interface, workload bounding

1. INTRODUCTION

Nowadays, the integration of tasks with different levels of criticality onto a shared hardware platform is becoming more and more important as this integration can increase the resource utilization and reduce the hardware cost. Particularly, in the traditional safety-critical avionics and automotive domains, the system is evolving into a mixed-criticality system (MCS) by which the stringent non-functional requirements w.r.t. cost, space, weight, heat disspation and power consumption must be met [Burns and Davis 2015]. For example, an unmanned aerial vehicle often integrates high-critical tasks, such as flight control, with low-critical tasks (like the mission tasks), together into a same processor.

The schedule of a MCS is recently priority-based, where the low-critical tasks are set with high priorities in order to improve their QoS. To guarantee the execution of the high-critical tasks, the high-priority low-critical tasks are constrained within a

This work has been partly funded by China Scholarship Council, German BMBF projects ECU (grant number: 13N11936), and China SYSU 'the Fundamental Research Funds for the Central Universities' (grant number: 15lgjc32).
Authors' addresses: K. Huang (corresponding author), School of Data and Computer Science, Sun Yat-sen University, Xiaoguwei Island, Panyu District, Guangzhou 510006, China. email: huangk36@mail.sys.edu.cn; B. Hu, L. Cheng, and A. Knoll, Technical University Munich, Boltzmannstrae 3, 85748, Garching, Germany; G. Chen, College of Computer Science and Engineering, Northeastern University (NEU), No 3-11, Wenhua Road, Heping District, Shenyang 110819, China.

DOI: 0000001.0000001

certain bound [Wandeler et al. 2012; Phan and Lee 2013; Tobuschat et al. 2014]. When their execution demand exceeds this bound, either the executions are delayed by a regulator [Wandeler et al. 2012] or the execution priorities are exchanged with the high-critical tasks [Tobuschat et al. 2014]. Nevertheless, computing such a bound for the low-critical tasks is nontrivial as the bound on the one hand should be sufficient to guarantee the execution of the high-critical tasks and on the other hand should not be too pessimistic to improve the QoS of low-critical tasks.

There is related work in the literature to offline compute a workload bound by which the incoming workload of the low-criticality is shaped at runtime. Wandeler *et al.* [Wandeler and Thiele 2005; Wandeler et al. 2012] proposed to use the Real-Time Interface to obtain the shaping bound and suggested to delay the non-real-time events when they exceed the precomputed bound. With the computed shaping bound, the work in [Neukirchner et al. 2012; Neukirchner et al. 2013b] discussed in more details on how to monitor the low-critical events by switching the workload bound distribution among low-critical tasks, or by using the workload arrival curve to monitor the whole group of low-critical events. Tobuschat *et al.* [Tobuschat et al. 2014] presented a scheme that exploits the throughput and latency slack of critical applications, by prioritizing non-critical accesses over critical accesses and exchanging the priorities when the non-critical workload exceeds the pre-computed bound.

The feasible bounding parameters in the aforementioned related work are obtained offline, i.e., the bound used to shape the workload of the low-critical tasks is computed offline and fixed during the runtime. This bound is computed based on the assumptions of the worst-case event arrival patterns of all high-critical tasks. While using the worst-case assumption during the runtime can guarantee the safeness of the workload bound, it also introduces the pessimism due to the differences between the actual demand and the assumed worst-case demand of the high-critical tasks. Such pessimism will significantly hamper the QoS of low-critical tasks and reduce the overall system utilization. To reduce the pessimism, the shaping bound should be adaptively computed at runtime based on the actual demand from the high-critical tasks. Such online adaption is however not so easy. On the one hand, the actual demand should be a valid upper bound that guarantees no high-critical tasks miss their deadlines. On the other hand, the adaption decisions should be lightweight. While most of the previous work relies on the heavy numerical computation, to the best of our knowledge, no work so far has considered adaptively refining the feasible workload bound at runtime.

Being aware of this, this article proposes to adaptively refine the bound of constraining the low-critical tasks at runtime. By using historical knowledge of the arrival workload in the past, the feasible bound for the low-critical tasks is dynamically refined. Based on the refined bound, the priority-adjustment policy and workload-shaping policy are proposed to manage the low-critical workload in order to provide the service for low-critical tasks as much as possible, while guaranteeing sufficient service for meeting the hard real-time constraints of high-critical tasks. The detail contributions of this article are as follows:

— We present an adaptive scheme for the online bound refinement in MCSs. By monitoring the arrival events of high-critical tasks, the actual demand is adaptively computed to shape the workload of the low-critical tasks at runtime. The computed actual demand can guarantee that the high-critical tasks meet all their deadlines, while at the same time increase the QoS of low-critical tasks.
— Two online workload management policies, namely, priority-adjustment policy and workload-shaping policy, are investigated. The priority-adjustment policy reduces the interference on high-critical tasks by decreasing the priority of low-critical tasks when the low-critical workload exceeds the refined bound. The workload-shaping

policy shapes the incoming low-critical workload to comply with the refined bound by keeping the priority of low-critical tasks always the highest.

— To update the refined bound for low-critical tasks, Real-Time Interface is traditionally used [Wandeler and Thiele 2005; Wandeler et al. 2012] to compute the provided service and demand bound functions for lower priority tasks. To eliminate the complex Real-Time Interface computation that requires intensive numerical calculation, a lightweight method with the complexity of $O(n \cdot log(n))$ is developed, making the online adaptations applicable at runtime.

— Extensive experiments are presented to show the efficiency and effectiveness of our two proposed workload management policies. Comparing with the method that applies the exact Real-Time Interface computation, the timing overhead of our proposed lightweight method is one and two orders of magnitude lower in the priority-adjustment policy and in the workload-shaping policy, respectively. The workload management effect of using the lightweight method has minor difference with that of using the exact Real-Time Interface method.

This article builds on the work presented in [Hu et al. 2015]. In the present version, this adaptive scheme is stated with more explanations and proofs. Except the shaping approach in [Hu et al. 2015], we also present the priority-adjustment policy in adaptively managing the workload. Besides, in this article the shaping behavior is different. In [Hu et al. 2015], the bound of constraining low-critical tasks is updated only when there are no pending high-critical events, because the model in [Hu et al. 2015] does not cover the case that there are some carry-on events (released but not finished). This article models the carry-on events, which makes it possible to update the bound anytime. Furthermore, more experiments are presented to show the effectiveness of our two proposed workload management policies.

The rest of this article is organized as follows: Section 2 reviews related work. Section 3 provides our system model and settings. Section 4 presents the schedulability analysis by applying the Real-Time Interface and the future workload bound by applying dynamic counters. Section 5 presents the priority-adjustment policy and workload-shaping policy to do the low-critical workload management. Section 6 proposes a lightweight method for refining the runtime workload bound. Simulation results are presented in Section 7 and Section 8 concludes this article.

2. RELATED WORK

Since the first paper on the verification of a proposed MCS in 2007 [Vestal 2007], improving the performance of MCS has been widely studied.

In contrast with the conventional real-time systems, a key aspect of the MCS is that system parameters, such as tasks' worst-case execution times (WCETs), become dependent on the criticality level of the tasks. Based on this model, a lot of approaches were proposed to improve the performance of MCS. For fixed-priority scheduling systems, Audsley's algorithm [Audsley 2001] was proved to be optimal to assign priorities to tasks in MCSs in [Dorin et al. 2010]. A more effective approach of response time analysis was proposed to improve the system schedulability in [Baruah et al. 2011b]. Regarding to the problem of unpractical model assumed in [Baruah et al. 2011b], more practical concerns were addressed in [Burns and Baruah 2013]. For the earliest-deadline-first scheduling system, EDF-VD (EDF-with virtual deadlines) [Baruah et al. 2011a] was proposed to successfully schedule mixed-critical tasks. An Elastic Mixed-Criticality task model was presented in [Su and Zhu 2013] to improve the service of low-critical tasks by allowing low-critical tasks to run more frequently in certain circumstances, and simulation results showed that the proposed Early-Release EDF algorithm under this model can schedule much more task sets than EDF-VD. The Elastic Mixed-Criticality task model was extended in [Su et al. 2014] to allow each low-critical task to have a pair of periods. Besides, virtual deadlines of

high-critical tasks and demand bound function analysis were introduced to explore the service guarantees to low-critical tasks in [Su et al. 2014].

All of the aforementioned works are based on the standard mixed-criticality model [Burns and Davis 2015], in which all tasks are sporadically activated and the system has a mode switch at runtime. In contrast with this standard model, the task model in this article is allowed to be activated with any pattern and the system has no mode switch at runtime. This model has been studied in recent work, such as [Neukirchner et al. 2013a; Neukirchner et al. 2013b; Tobuschat et al. 2014]. In the following, we review the existing workload management approaches under this task model.

In the network calculus [Le Boudec and Thiran 2001], the greedy shapers were proposed to strictly conform the overloaded packets to the arrival constraints. An important property of the greedy shaper is "greedy shaper comes for free" [Le Boudec and Thiran 2001], which means that the shaping does not increase delay or buffer requirements. The use of greedy shaper was introduced to schedule the real-time events in [Wandeler et al. 2012], and it was found that the property of "greedy shaper comes for free" is also applicable for shaping real-time events. In [Dewan and Fisher 2012], an admission server was proposed to enforce arbitrary real-time demand-curve interfaces. However, its implementation overhead is not low. In [Huang et al. 2012], by implementing a dual-bucket mechanism (similar to dynamic counters [Lampka et al. 2011]) into FPGA, the runtime inputs are conformed to the designed arrival constraints. The implementation showed that the resource and timing overheads of this shaping scheme are very low in FPGA. Another monitoring method based on the minimum distance functions (i.e., an inverse representation of arrival curves) was proposed to do the event model verification [Neukirchner et al. 2012]. This method is able to monitor the periodic burst events. Its runtime overhead can be reduced by using an l-repetitive function to represent the minimum distance function. The monitoring difference between using the dynamic counters and l-repetitive function was explored in [Hu et al. 2015; Hu et al. 2016]. Although both dynamic counters and l-repetitive functions are effective in monitoring some certain-pattern event traces, only dynamic counters approach was presented how to use the past monitoring results to predict the future event traces [Lampka et al. 2011].

The multi-mode monitoring and workload monitoring were proposed in [Neukirchner et al. 2013b] to monitor the low-critical workload by switching the workload bound distribution of specific low-critical tasks. A similar approach is to monitor the low-critical events as a group [Neukirchner et al. 2013a] by using workload arrival curve, in which way the system utilization increases while the timing constraints of high-critical tasks are met. By monitoring the runtime workload, a scheme named workload-aware shaping was proposed to improve the resource utilization by prioritizing the low-critical accesses over high-critical accesses, and exchange the priorities among them only when the incoming low-critical workload exceeds the designed bound [Tobuschat et al. 2014]. Phan *et al.* [Phan and Lee 2013] introduced another technique to use an optimal greedy shaper for shaping periodic tasks with jitter to improve the schedulability of real-time systems.

None of the preceding monitoring or shaping scheme allows to refine the shaping bound. Thus, their shaping schemes are pessimistic because of the differences between the actual demand and the worst-case demand. In this article, we aim to fill this gap by providing an adaptive bound refinement scheme.

3. SYSTEM MODEL AND SETTINGS

3.1. System Model

3.1.1. Event Stream Model. In this article, the job of a task is assumed to be activated by an input event, which is a common assumption in real-time systems. Task activations

in the system are thus expressed as an event stream. A trace of such an event stream is described by means of a differential arrival function $R[s,t)$ that denotes the sum of events arrived in the time interval $[s,t)$, with $R[s,s)=0, \forall s,t \in \mathbb{R}$. While any R always describes one concrete trace, a 2-tuple $\overline{\alpha}(\Delta) = [\overline{\alpha}^u(\Delta), \overline{\alpha}^l(\Delta)]$ provides an upper and lower bound on the number of events over any time interval of length Δ.

Definition 3.1 (*Event Arrival Curve [Wandeler 2006]*). *Denote $R[s,t)$ as the number of events that arrive on an event stream in the time interval $[s,t)$. Then, $\overline{\alpha}^u$ and $\overline{\alpha}^l$ represent the upper and lower bound on the number of event in any interval $t-s$, that is,*

$$\overline{\alpha}^l(t-s) \leq R[s,t) \leq \overline{\alpha}^u(t-s), \forall t \geq s \geq 0,$$

with $\overline{\alpha}^l(\Delta) \geq 0$, $\overline{\alpha}^u(\Delta) \geq 0$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$.

Especially, for a pjd event stream with period p, jitter j, and minimal inter arrival distance d, the upper event arrival curve is that

$$\overline{\alpha}^u(\Delta) = \min\{\lceil \frac{\Delta + j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}. \tag{1}$$

For the easiness, to represent the upper event arrival curve of task τ_i, we denote it as $\overline{\alpha}^u(\tau_i, \Delta)$. Similar denotation are used in the following.

3.1.2. Resource Model. Analogous to the event arrival curve that provides an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curve then provides an abstract resource model.

Definition 3.2 (*Service Curve [Wandeler 2006]*). *Denote $C[s,t)$ as the number of events that arrive on an event stream in the time interval $[s,t)$. Then, β^u and β^l represent the upper and lower bound on the resource availability in any interval $t-s$, that is,*

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s), \forall t \geq s \geq 0,$$

with $\beta^l(\Delta) \geq 0$, $\beta^u(\Delta) \geq 0$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$.

Again, service curves substantially generalize classical resource models, such as the bounded delay or the periodic resource model. Besides, $\beta^l(\tau_i, \Delta)$ denotes the minimum service that a task τ_i obtains on the resource over any time interval Δ.

3.1.3. Workload Model. As an event arrival curve $\overline{\alpha}$ specifies the event and a service curve β specifies the available processing time, the event arrival curve $\overline{\alpha}$ has to be transformed to the workload arrival curve α to indicate the amount of computation time required for the arrived events in intervals.

Definition 3.3 (*Workload Arrival Curve [Wandeler 2006]*). *Denote $W[s,t)$ as the number of clock cycles required to process $t-s$ consecutive events on a computer or communication resource. Then, α^u and α^l represent the upper and lower bound on the required cycles in any interval $t-s$, that is,*

$$\alpha^l(t-s) \leq W[s,t) \leq \alpha^u(t-s), \forall t \geq s \geq 0,$$

with $\alpha^l(\Delta) \geq 0$, $\alpha^u(\Delta) \geq 0$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$.

Suppose that the WCET of an event stream is c, then the transformation can be done by $\alpha^u = c \cdot \overline{\alpha}^u$, $\alpha^l = c \cdot \overline{\alpha}^l$ and back by $\overline{\alpha}^u = \alpha^u / c$, $\overline{\alpha}^l = \alpha^l / c$. For a task τ_i, the upper bound workload for it can be denoted as $\alpha^u(\tau_i, \Delta)$.

3.1.4. Demand Bound Function. In order to analyze the schedulability of real-time tasks, a easy approach is to use demand bound functions.

Definition 3.4 (*Demand Bound Function [Ekberg and Yi 2012]*). *A demand bound function* $\mathrm{dbf}(\tau_i, \Delta)$ *gives an upper bound on the maximum possible execution demand of task* τ_i *in any time interval of length* Δ*, where demand is calculated as the total amount of required execution time of jobs with their whole scheduling windows within the time interval.*

Based on this definition, a tight demand bound function can be obtained by the following lemma.

LEMMA 3.5 ([THIELE ET AL. 2006]). *Suppose a task* τ_i *with relative deadline* D_i *and upper arrival curve* $\alpha(\tau_i, \Delta)$*. To satisfy the required relative deadline* D_i*, the demand bound function of* τ_i *is*

$$\mathrm{dbf}(\tau_i, \Delta) = \alpha^u(\tau_i, \Delta - D_i). \tag{2}$$

The key point that makes demand and supply bound functions useful for the schedulability analysis of real-time systems is the following fact.

PROPOSITION 3.6 ([THIELE ET AL. 2000]). *The task* τ_i *is schedulable if and only if the condition*

$$\beta^l(\tau_i, \Delta) \geq \mathrm{dbf}(\tau_i, \Delta) \tag{3}$$

holds.

3.1.5. Processing Model. In the framework of Real-Time Calculus, the task processing is often modeled by abstract performance component that acts as curve transformer in the domain of arrival and service curve, where the transferring function depends on the modeled processing semantics. The Greedy Processing Component (GPC) models a task that is triggered by the events which are queued up in the FIFO (first-in-first-out) buffer. A typical case is shown in Fig. 1, where $[\alpha_i^u, \alpha_i^l]$ and $[\beta_i^u, \beta_i^l]$ are respectively the workload arrival curve and resource service curve. The processing of two tasks in a preemptive fixed-priority scheduling system is abstracted as two GPCs. The lower service for the lower-priority task is the processing service left over after processing the higher-priority task [Wandeler 2006]:

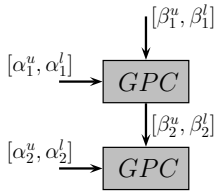


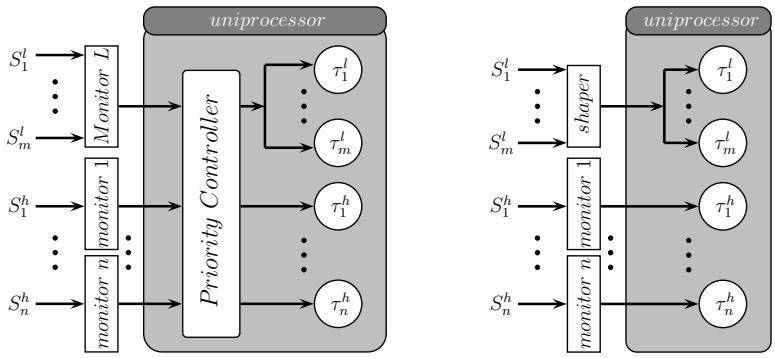
Fig. 1. Processing model of two tasks scheduled by fixed priority

$$\beta_2^l(\Delta) = \mathrm{RT}(\beta_1^l, \alpha_1^u)(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\beta_1^l(\lambda) - \alpha_1^u(\lambda)\}. \tag{4}$$

3.2. System Settings

This article considers a uniprocessor that is scheduled according to the preemptive fixed-priority (FP) scheduling policy. Two system settings w.r.t. the two workload management policies are presented, as shown in Fig. 2. In both systems, there are a set of low-critical tasks $\tau^l = \{\tau_1^l, \tau_2^l, ..., \tau_m^l\}$ and a set of high-critical tasks $\tau^h = \{\tau_1^h, \tau_2^h, ..., \tau_n^h\}$. The corresponding event streams for them are denoted as $S^l = \{S_1^l, S_2^l, ..., S_m^l\}$ and $S^h = \{S_1^h, S_2^h, ..., S_n^h\}$. For every high-critical event stream, a monitor is applied to monitor its arrival events. For the high-critical tasks, in order to strictly meet their deadlines, their activation patterns are often fixed. Therefore, their lower and upper arrival curves are known in the system design-time stage. For low-critical tasks, such as multi-media tasks in a car, their workload may sometimes be overloaded. Hence, their workload needs to be regulated, in order to constrain their interferences on high-critical tasks. This system assumption is the same as recent works of [Neukirchner et al. 2013a; Neukirchner et al. 2013b; Tobuschat et al. 2014].

In the case for priority-adjustment policy, as shown in Fig. 2(a), a priority controller is applied in the system to dynamically assign priorities for the arriving events. The low-critical events are grouped together sharing the same priority, and are monitored



(a) System setup for priority-adjustment policy

(b) System setup for workload-shaping policy

Fig. 2. Mixed-criticality systems scheduled by the preemptive FP policy

by the Monitor L during the runtime. The priority of all low-critical tasks is named low-critical priority in the following. The priorities order of high-critical tasks is fixed and unchanged during the runtime, while the low-critical priority will be changed to lower level or higher level, subjected to the actual demand of high-critical tasks. This means that the priority controller can only change the priority of τ^l from 1 to $n+1$. Without the loss of generality, the tasks $\tau_1^h, ..., \tau_n^h$ in τ^h are prioritized in a descending order, that is, the priority of the task τ_i^h is higher than that of the task τ_j^h when $i < j$.

In the case for workload-shaping policy, as shown in Fig. 2(b), a shaper is used to manage the inflowing workload from low-critical event streams. In this case, the priorities of all tasks cannot be changed during the runtime. The priorities for high-critical tasks $\tau_1^h, ..., \tau_n^h$ are also set as a descending order. All low-critical tasks are grouped together to share the highest priority among all tasks, with the aim of maximizing the service for low-critical tasks. In contrast to the priority controller in priority-adjustment policy, the shaper is only responsible for the release of low-critical events. The principle of releasing an event is that, the released event should not result in a deadline miss of high-critical tasks.

In all monitors or shapers, buffers are used to store the backlogged events for every event stream, because there may be some events that have arrived but not released. The buffer obeys the principle of first-come-first-serve. The size of the buffer is assumed to be large enough. The effect of buffer size on the workload management is out of the scope of this article. Note that, although the WCET of every event can be different, the low-critical events can be simply considered from one stream whose workload is bounded by a workload arrival curve, which is similar to the workload bound of a group tasks in [Neukirchner et al. 2013a].

4. REAL-TIME CALCULUS ROUTINES AND INTERFACE ANALYSIS

In this section, we present the basic routines to construct the actual arrival curve and demand bound function at runtime, on top of which we present the interface analysis to guarantee no deadline misses for high-critical tasks.

4.1. Arrival Curves and Demand Bound Functions with Historical Information

Before presenting the schedulability analysis, we first introduce how to derive the actual arrival curve and demand bound functions with historical information. The actual arrival curve consists of the workload of future events, the backlogged events, and the carry-on event (active and not finished), while the demand bound function also consists of the demand of the future events, the backlogged events, and the carry-on event.

Algorithm 1 Implement a dynamic counter to track a staircase function

Input: signal s, $\triangleright tuple < DC_j, CLK_j >$;

1: **if** s = eventArrival **then**
2: **if** $DC_j = N_j^u$ **then**
3: reset_timer(CLK_j, δ_j^u)
4: $k_j = 0$
5: **end if**
6: $DC_j \leftarrow DC_j - 1$
7: **end if**
8: **if** $s = CLK_j Timeout$ **then**
9: $DC_j \leftarrow \min(DC_j{+}1, N_j^u)$
10: reset_timer(CLK_j, δ_j^u)
11: $k_j = k_j + 1$
12: **end if**
13: **if** $DC_j < 0$ **then**
14: report_exception
15: **end if**

4.1.1. Future Events and their Demand Bound. The arrival pattern of future events can be predicted by using the past information. We adopt the dynamic counters approach to monitor the runtime events as this approach was presented how to fast obtain a tight bound on the number of future events.

In principle, any (discrete) complex arrival pattern can be bounded by a set of upper and lower staircase functions [Lampka et al. 2009]. Therefore, suppose the arrival curve $\overline{\alpha}^u(\tau_i, \Delta)$ of task τ_i is composed of n' staircase functions, i.e.,

$$g\forall\Delta \in \mathbb{R}_{\geq 0} : \overline{\alpha}^u(\tau_i, \Delta) \leq \min_{j=1..n'}\{N_j^u + \lfloor\frac{\Delta}{\delta_j^u}\rfloor\},$$

where N_j^u is the initial value of a staircase function and δ_j^u is the stair length.

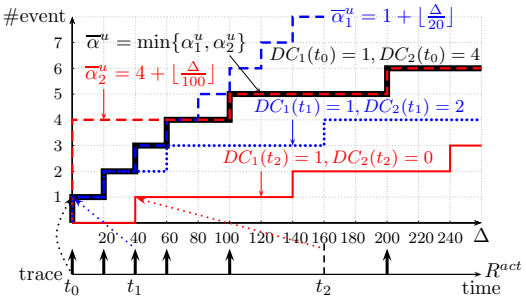


Fig. 3. An example for using dynamic counters to predict the future events

A dynamic counter (DC_j) can be used to track a single upper staircase function ($\overline{\alpha}_j^u$) and predict its future workload. The detailed tracking algorithm can be seen in Algo. 1 [Lampka et al. 2011]. DC_j tracks the potential burst capacity, and the auxiliary variable k_j in Algo. 1 tracks the offset between the current time t and the last δ_j^u. Below shows an example of using dynamic counters for event prediction.

Example 4.1. *As shown in Fig. 3, for the PJD task with (P, J, D)=(100, 300, 20), two staircase functions, $\overline{\alpha}_1$ and $\overline{\alpha}_2$, are used to approximate the arrival curve of this PJD task. Every staircase function is tracked by a counter. Assume the real arrival event trace is shown in the event trace R^{act}. By applying Algo. 1, DC_1 tracks the arrival event trace based on $\overline{\alpha}_1$, and DC_2 tracks the arrival event trace based on $\overline{\alpha}_2$. The minimum of $DC_1(t)$ and $DC_2(t)$ is the potential activations of this task at time t.*

As shown in Fig. 3, the bold line shows the worst-case arrival pattern at the beginning, $DC_1(t_0) = N_1^u = 1$, $DC_2(t_0) = N_2^u = 4$. At time t_1, two events have been recorded, and dynamic counters are updated to that $DC_1(t_1) = 1$, $DC_2(t_1) = 2$. The future event prediction is shown in the dotted line, which is less than the worst-case assumption. At time t_2, dynamic counters are updated to that $DC_1(t_2) = 1$, $DC_2(t_2) = 0$ since five events arrived during $[t_0, t_2]$. The prediction shown in the solid line is further less than the one at time t_1.

The potential burst capacity $DC_j(t)$, together with staircase function, yields the following future events prediction [Lampka et al. 2011]:

$$\mathcal{U}_j(\tau_i, \Delta, t) = DC_j(t) + \begin{cases} \lfloor\frac{\Delta+(t-k_j\delta_j^u)}{\delta_j^u}\rfloor & \text{if } DC_j(t) < N_j^u \\ \lfloor\frac{\Delta}{\delta_j^u}\rfloor & \text{if } DC_j(t) = N_j^u \end{cases} \quad (5)$$

The above function bounds future event arrivals at time t. By using the monitor to track the event trace for tasks, DC_j and k_j are known during the runtime. Therefore, $\mathcal{U}_j$ is also known. For bounding the number of future event arrivals w.r.t. a complex activations pattern of task τ_i, one can simply take the minimum over all the $\mathcal{U}_j$ [Lampka et al. 2011]:

$$\overline{\alpha}^u(\tau_i, \Delta, t) = \min_{j=1..n'}(\mathcal{U}_j(\tau_i, \Delta, t)). \tag{6}$$

From lemma 3.5, it is derived that the demand bound function of future events are that

$$\mathrm{dbf}^{\mathrm{F}}(\tau_i, \Delta, t) = \overline{\alpha}^u(\tau_i, \Delta - D_i, t) \cdot c_i, \tag{7}$$

where D_i and c_i are the relative deadline and WCET of task τ_i.

4.1.2. Backlogged Events and their Demand Bound. During the runtime, if events arrive more frequently than the rate that they can be processed, some events may be backlogged. We denote the set of unfinished events of τ_i in the backlog at time t as $\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. Then, the number of backlogged events can be denoted as $|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$. For each event $e_{i,j} \in \mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$, we use $D_{i,j}$ to denote its absolute deadline.

Definition 4.2 (*Backlogged Demand [Huang et al. 2011]*). *Suppose the set of unfinished events of a task τ_i in the buffer at time t are denoted as $\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. Let $D_{i,j}$ denote the absolute deadline for event $e_{i,j} \in \mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. A backlogged demand for this task is defined as*

$$\mathrm{dbf}^{\mathrm{B}}(\tau_i, \Delta, t) = c_i \cdot \begin{cases} (j-1), & D_{i,j} - t < \Delta < D_{i,j+1} - t, \\ |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|, & \Delta \geq D_{i,|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|} - t, \end{cases}$$

in which, c_i is the WCET*, and $D_{i,0}$ is defined as t for brevity.*

4.1.3. Carry-On Event and its Demand Bound. Suppose at time t, an event of task τ_i has been released but its execution is not finished. This event is called a carry-on event. Similar to the demand of backlogged events, the demand of carry-on event is defined as follows.

Definition 4.3 (*Carry-On Demand*). *Suppose $C(\tau_i, t)$ is used to denote the left time for finishing a carry-on event of τ_i at time t, and the demand for the carry-on event is that*

$$\mathrm{dbf}^{\mathrm{C}}(\tau_i, \Delta, t) = c_i \cdot \begin{cases} 0, & \Delta < D_c - t, \\ C(\tau_i, t), & \Delta \geq D_c - t, \end{cases}$$

where D_c is the absolute deadline of this carry-on event.

Based on the concept of arrival curve and demand bound function, it can be known that the workload arrival curve $\alpha^u(\tau_i, \Delta, t)$ and demand bound function $\mathrm{dbf}(\tau_i, \Delta, t)$ of task τ_i are that,

$$\begin{aligned} \alpha^u(\tau_i, \Delta, t) &= c_i \cdot \overline{\alpha}^u(\tau_i, \Delta, t) + c_i \cdot |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})| + C(\tau_i, t), \\ \mathrm{dbf}(\tau_i, \Delta, t) &= \mathrm{dbf}^{\mathrm{F}}(\tau_i, \Delta, t) + \mathrm{dbf}^{\mathrm{B}}(\tau_i, \Delta, t) + \mathrm{dbf}^{\mathrm{C}}(\tau_i, \Delta, t). \end{aligned} \tag{8}$$

Note that, since future events, backlogged events, and carry-on event are known at runtime, the actual arrival curve and demand bound function are also known at runtime.

4.2. Schedulability Analysis Based on Real-Time Interface

Based on the proposition 3.6, to sufficiently guarantee that the task τ_i can meet its deadline from time t, the provided lower service should be greater than its demand bound function, i.e.,

$$\beta^l(\tau_i, \Delta, t) \geq \mathrm{dbf}(\tau_i, \Delta, t). \tag{9}$$

In the following, two approaches are presented to analyze the system schedulability. One is to derive the whole demand bound function of all high-critical tasks, and guarantee that the provided lower service is greater than the whole demand bound function. This approach is often used in deriving the bound offline [Wandeler et al. 2012; Neukirchner et al. 2013a; Neukirchner et al. 2013b; Tobuschat et al. 2014]. The other approach is to guarantee the task schedulability individually, in which there are n inequalities to be guaranteed.

4.2.1. Schedulability Analsyis by Considering High-Critical Tasks as a Group. In the initial setting, as the priority of low-critical tasks is set as the highest, the system can be abstracted as Fig. 4, where

— $\alpha^u(\tau^l, \Delta, t)$ denotes the workload arrival curve of the set of all low-critical tasks,
— $\beta^l(\tau^l + \tau^h, \Delta, t)$ denotes the provided lower service curve for all tasks,
— $\beta^l(\tau^h, \Delta, t)$ denotes the provided service for the set of all high-critical tasks,
— and $\mathrm{dbf}(\tau^h, \Delta, t)$ denotes the demand bound function to meet deadlines of all high-critical tasks.

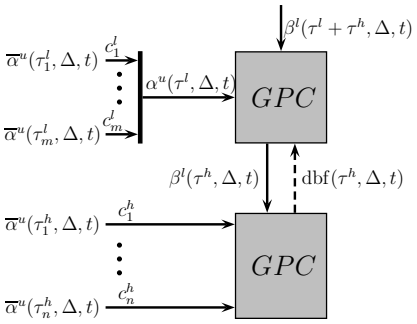


Fig. 4. Real-Time Interface analysis

Furthermore, for the easiness of the analysis in the following, we denote that

— $\tau_i^{h,n}$ represents the set of high-critical tasks of τ_i^h, τ_{i+1}^h, ..., τ_n^h. Then, we have $\tau_1^{h,n} = \tau^h$. By this way, $\beta^l(\tau_i^{h,n}, \Delta, t)$ and $\mathrm{dbf}(\tau_i^{h,n}, \Delta, t)$ respectively represent the lower service curve and the demand bound function for high-critical tasks from τ_i^h to τ_n^h.

THEOREM 4.4. *In order to guarantee that all high-critical tasks are schedulable, the maximum feasible* $\alpha^u(\tau^l, \Delta, t)$ *is that,*

$$\alpha^u(\tau^l, \Delta, t) = \mathrm{RT}^{-\alpha}\Big(\mathrm{dbf}(\tau^h, \Delta, t),\ \beta^l(\tau^l + \tau^h, \Delta, t)\Big)^1. \tag{10}$$

PROOF. To sufficiently guarantee that all high-critical tasks can be scheduled, the provided service for the set of high-critical tasks should be greater than their demand bound function, i.e.,

$$\beta^l(\tau^h, \Delta, t) \geq \mathrm{dbf}(\tau^h, \Delta, t). \tag{11}$$

According to the processing model of fixed-priority, we have

$$\beta^l(\tau^h, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau^l + \tau^h, \Delta, t), \alpha^u(\tau^l, \Delta, t)\Big). \tag{12}$$

By inverting Eq. 12, we can get that,

$$\alpha^u(\tau^l, \Delta, t) = \mathrm{RT}^{-\alpha}\Big(\beta^l(\tau^h, \Delta, t),\ \beta^l(\tau^l + \tau^h, \Delta, t)\Big).$$

Since $\beta^l(\tau^h, \Delta, t) \geq \mathrm{dbf}(\tau^h, \Delta, t)$, the maximum feasible $\alpha^u(\tau^l, \Delta, t)$ is obtained by using $\mathrm{dbf}(\tau^h, \Delta, t)$ to replace $\beta^l(\tau^h, \Delta, t)$. Hence, Eq. 10 holds. □

[1] $\mathrm{RT}^{-\alpha}(\beta', \beta)(\Delta) = \beta(\Delta + \lambda) - \beta'(\Delta + \lambda)$ for $\lambda = \sup\{\tau : \beta'(\Delta + \tau) = \beta'(\Delta)\}$, from the real-time interface in [Wandeler and Thiele 2005].

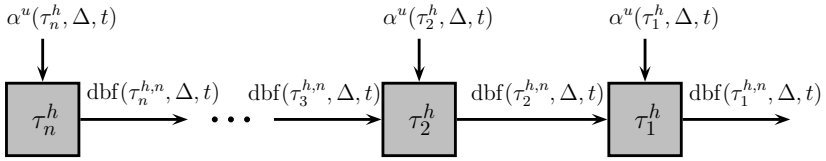


Fig. 5. The flow of backward derivation

In Eq. 10, $\beta^l(\tau^l + \tau^h, \Delta, t)$ is also the full processing ability of the platform. As the platform is assumed to be a uniprocessor with constant processing ability, $\beta^l(\tau^l + \tau^h, \Delta, t) = \Delta$ during the runtime. Then, to get $\alpha^u(\tau^l, \Delta, t)$, one has to know $\mathrm{dbf}(\tau^h, \Delta, t)$. Based on the Real-Time Interface, $\mathrm{dbf}(\tau^h, \Delta, t)$ can be known by the backward derivation [Wandeler and Thiele 2005; Thiele et al. 2006]. The backward derivation step to get $\mathrm{dbf}(\tau^h, \Delta, t)$ is briefly introduced as follows.

As shown in Fig. 5, for achieving the schedulability of tasks from τ_i^h to τ_n^h, $\mathrm{dbf}(\tau_i^{h,n}, \Delta, t)$ should be greater than the demand bound function of task τ_i^h, and the provided service for tasks from τ_{i+1}^h to τ_n^h should be greater than $\mathrm{dbf}(\tau_{i+1}^{h,n}, \Delta, t)$, i.e.,

$$\begin{aligned} \mathrm{dbf}(\tau_i^{h,n}, \Delta, t) &\geq \mathrm{dbf}(\tau_i^h, \Delta, t), \\ \mathrm{RT}\Big(\mathrm{dbf}(\tau_i^{h,n}, \Delta, t), \alpha^u(\tau_i^h, \Delta, t)\Big) &= \beta(\tau_{i+1}^{h,n}, \Delta, t) \geq \mathrm{dbf}(\tau_{i+1}^{h,n}, \Delta, t). \end{aligned} \tag{13}$$

Therefore, by inverting Eq. 13, we can get that

$$\mathrm{dbf}(\tau_i^{h,n}, \Delta, t) = \max\Big\{\mathrm{dbf}(\tau_i^h, \Delta, t), \mathrm{RT}^{-\beta}\Big(\mathrm{dbf}(\tau_{i+1}^{h,n}, \Delta, t), \alpha^u(\tau_i^h, \Delta, t)\Big)\Big\}^2. \tag{14}$$

By applying Eq. 14 for $i = n-1, n-2, ..., 1$, the demand bound function $\mathrm{dbf}(\tau_1^{h,n}, \Delta, t)$ for the set of all high-critical tasks is derived. Then, by applying Eq. 10, the maximum workload arrival curve $\alpha^u(\tau^l, \Delta, t)$ for low-critical tasks can be computed.

4.2.2. Schedulability Analysis by Considering High-Critical Tasks Separately. By considering the schedulability of each high-critical task, we have the following theorem.

THEOREM 4.5. *All high-critical tasks are schedulable if and only if*

$$\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t), \ \forall\, i \in [1..n]. \tag{15}$$

PROOF. This theorem directly stems from the schedulable condition of Eq. 9. □

5. MANAGEMENT POLICY ON LOW-CRITICAL TASKS

By constantly updating the upper bound of constraint on low-critical tasks, one can prevent the low-critical interferences on high-critical tasks either by decreasing their execution priorities or by using a shaper to shape the overloaded workload when the low-critical workload violates the bound. In this section, we present how to apply the priority-adjustment policy and workload-shaping policy in managing the low-critical workload at runtime.

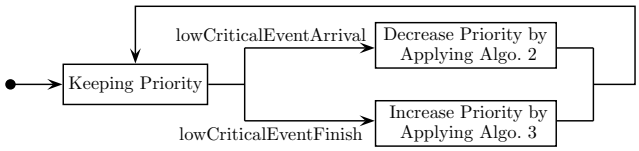


Fig. 6. The flow of priority-adjustment policy

5.1. Priority-Adjustment Policy

For the priority-adjustment policy, since the relative priority order of high-critical tasks is fixed, we have to decide when to decrease or increase the priority of low-critical

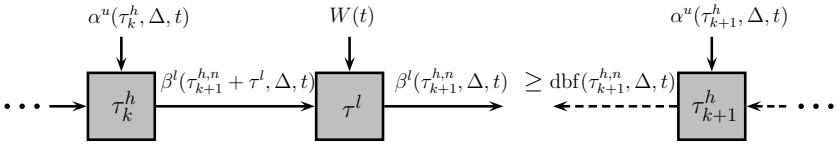


Fig. 7. The diagram showing the verification of system scheduliability by priority-adjustment policy

tasks. All low-critical tasks share one priority, which indicates that they are scheduled as a group. In the initial setting, the priority of low-critical tasks is set as the highest. During the runtime, their priority is dynamically adjusted based on the actual demand bound functions of high-critical tasks. An overview of the priority-adjustment policy is illustrated in Fig. 6. Whenever a low-critical event arrives or a low-critical event is finished, the low-critical priority will be decreased or increased, in order to keep no deadline miss of all high-critical tasks.

5.1.1. Decreasing Priority. For the decrease of the priority of all low-critical tasks, we have to verify whether current low-critical workload has already violated the bound. Since the low-critical workload is unpredictable, its future bound is unknown. Such verification only relies on the low-critical workload that has already arrived. Therefore, every time when a new low-critical event arrives, the schedulability verification should be done to guarantee that all high-critical tasks can be scheduled. If the new arrival event results in other high-critical tasks missing their deadlines, the low-critical priority should be decreased to avoid the deadline miss. Now we formulate a general statement for the problem of how to find a feasible priority for low-critical tasks.

Problem. *Let $P(\tau^l)$ denote the priority of the set of low-critical tasks and $P(\tau_i^h)$ denote the priority of the high-critical task τ_i^h. Suppose before time t, all high-critical tasks are schedualable with the priorities order that $P(\tau_1^h) > ... > P(\tau_k^h) > P(\tau^l) > P(\tau_{k+1}^h) > ... > P(\tau_n^h)$. If a new low-critical event arrives at time t, how to assign the priorities so that all high-critical tasks can still keep schedulable.*

The principle for reassigning the priorities is that, the schedulability of high-critical tasks should be guaranteed. The schedulability verification diagram is shown in Fig. 7. Suppose the low-critical workload becomes $W(t)$ with the arrival of a new event. Then, by using the backward derivation, the demand bound function $\mathrm{dbf}(\tau_{k+1}^{h,n}, \Delta, t)$ for serving tasks from τ_{k+1}^h to τ_n^h is derived. In order to meet the deadlines of tasks $\tau_{k+1}^{h,n}$, the provided service $\beta^l(\tau_{k+1}^{h,n}, \Delta, t)$ for them should be greater than their demand bound function, that is,

$$\beta^l(\tau_{k+1}^{h,n}, \Delta, t) \geq \mathrm{dbf}(\tau_{k+1}^{h,n}, \Delta, t). \tag{16}$$

By applying the forward computation of RT, we get the provided service for tasks $\tau_{k+1}^{h,n}$ and τ^l, which is,

$$\beta^l(\tau_i^{h,n} + \tau^l, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau_{i-1}^{h,n} + \tau^l, \Delta, t), \alpha^u(\tau_{i-1}^h, \Delta, t)\Big), \tag{17}$$

by applying $i = 2, ..., k+1$ and $\beta^l(\tau^h + \tau^l, \Delta, t) = \Delta$. To get $\beta^l(\tau_{k+1}^{h,n}, \Delta, t)$, the low-critical workload should also be considered because the priority of low-critical tasks is greater than tasks $\tau_{k+1}^{h,n}$. That is,

$$\beta^l(\tau_{k+1}^{h,n}, \Delta, t) = \beta^l(\tau_{k+1}^{h,n} + \tau^l, \Delta, t) - W(t). \tag{18}$$

If Eq. 16 does not hold, the low-critical priority has to be decreased. In general, the Algo. 2 can be applied to find the highest feasible priority for low-critical tasks. The procedure is that, every priority that is equal to or lower than the original low-critical priority is a possible priority for low-critical tasks. Therefore, Algo. 2 checks every

Algorithm 2 Procedure of priority reassignment with a new arrival low-critical event

1: **for** $k^\flat = k+1 \to n$ **do**
2: Compute $\beta^l(\tau^{h,n}_{k^\flat+1}, \Delta, t)$ and $\mathrm{dbf}(\tau^{h,n}_{k^\flat+1}, \Delta, t)$ by Eqs. 14, 17, 18;
3: **if** $\beta^l(\tau^{h,n}_{k^\flat+1}, \Delta, t) \geq \mathrm{dbf}(\tau^{h,n}_{k^\flat+1}, \Delta, t)$ **then**
4: Change the priorities order to be that $P(\tau^h_1) > ... > P(\tau^h_{k^\flat}) > P(\tau^l) > P(\tau^h_{k^\flat+1}) > ... > P(\tau^h_n)$;
5: break;
6: **end if**
7: **end for**

possible priority from high to low until a feasible priority is found out. This algorithm leads to the following theorem.

THEOREM 5.1. *Algo. 2 can always find a schedule, i.e., a priority assigned to low-critical tasks, to sufficient guarantee all high-critical tasks schedulable.*

PROOF. Denote t^- as the time instant just before the arrival of new event, and denote t^+ as the time instant just after the arrival of new event. The demand bound functions of all high-critical tasks are the same at t^- and at t^+. Since all high-critical tasks are schedulable at time t^-, we have

$$\begin{aligned} \beta^l(\tau^h_i, \Delta, t^-) \geq \mathrm{dbf}(\tau^h_i, \Delta, t^-) = \mathrm{dbf}(\tau^h_i, \Delta, t^+),\ \forall\, i \in [1..k], \\ \beta^l(\tau^{h,n}_{k+1}, \Delta, t^-) \geq \mathrm{dbf}(\tau^{h,n}_{k+1}, \Delta, t^-) = \mathrm{dbf}(\tau^{h,n}_{k+1}, \Delta, t^+). \end{aligned} \tag{19}$$

Since the priorities of tasks from τ^h_1 to τ^h_k is greater than that of τ^l, the services for tasks from τ^h_1 to τ^h_k are not changed by the arrival of new low-critical event, i.e., $\beta^l(\tau^h_i, \Delta, t^-) = \beta^l(\tau^h_i, \Delta, t^+),\ \forall\, i \in [1..k]$. Hence, tasks from τ^h_1 to τ^h_k are still schedulable.

With the new arrival of low-critical event, the low-critical workload changes from $W(t^-)$ to be $W(t^+)$. The worst case of applying Algo. 2 is to assign the lowest priority to low-critical tasks. We now prove that the lowest priority is always a feasible priority.

Suppose the priority of low-critical tasks is the lowest at t^+, then the processing resource $\beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^+)$ will be fully provided to the task set $\tau^{h,n}_{k+1}$, and only the remaining service after processing $\tau^{h,n}_{k+1}$ can be provided to low-critical tasks. Therefore, we have

$$\beta^l(\tau^{h,n}_{k+1}, \Delta, t^+) = \beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^+) = \beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^-).$$

Because $\beta^l(\tau^{h,n}_{k+1}, \Delta, t^-) = \beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^-) - W(t^-)$, we have

$$\beta^l(\tau^{h,n}_{k+1}, \Delta, t^+) = \beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^-) \geq \beta^l(\tau^{h,n}_{k+1} + \tau^l, \Delta, t^-) - W(t^-) = \beta^l(\tau^{h,n}_{k+1}, \Delta, t^-)$$

Because of Eq. 19, we can get that $\beta^l(\tau^{h,n}_{k+1}, \Delta, t^+) \geq \mathrm{dbf}(\tau^{h,n}_{k+1}, \Delta, t^+)$. Hence, we have proved that the lowest priority is always a feasible priority to be assigned to low-critical tasks and all tasks in $\tau^{h,n}_{k+1}$ are also schedulable. □

5.1.2. Increasing Priority. In our policy, we suppose that the low-critical priority can be increased when the low-critical workload is decreased. It indicates that, whenever a low-critical event has been finished, the low-critical priority can be increased. This is because, the interference imposed on high-critical tasks with priorities lower than the low-critical priority also decreases. The algorithm of increasing the low-critical priority is similar to the algorithm of decreasing the low-critical priority. As shown in Algo. 3, Eq. 16 is used to guarantee that the increased priority will not result in a deadline miss of all lower-priority tasks. The chosen priority gradually increases until an infeasible priority is found out. The reason for gradually increasing the chosen priority in Algo. 3

Algorithm 3 Procedure of priority reassignment after finishing a low-critical event

1: **for** $k^\flat = k \to 1$ **do**
2: Compute $\beta^l(\tau^{h,n}_{k^\flat}, \Delta, t)$ and $\mathrm{dbf}(\tau^{h,n}_{k^\flat}, \Delta, t)$ by Eqs. 14, 17, 18;
3: **if** $\beta^l(\tau^{h,n}_{k^\flat}, \Delta, t) < \mathrm{dbf}(\tau^{h,n}_{k^\flat}, \Delta, t)$ **then**
4: Change the priorities order to be that $P(\tau^h_1) > ... > P(\tau^h_{k^\flat+1}) > P(\tau^l) > P(\tau^h_{k^\flat}) > ... > P(\tau^h_n)$;
5: break;
6: **end if**
7: **end for**

or gradually decreasing the chosen priority in Algo. 2, instead of searching the priority from the highest to the lowest in Algo. 3, or the lowest to the highest in Algo. 2, is that the low-critical priority is supposed to keep unchanged with the finish or the arrival of the low-critical event.

THEOREM 5.2. *Algo. 3 can always find a schedule, i.e., a priority assigned to low-critical tasks, to sufficient guarantee all high-critical tasks schedulable.*

We omit the proof due to the similarity to theorem 5.1. The worst case of applying Algo. 3 is to keep the low-critical priority unchanged.

5.1.3. Runtime Behavior. The monitors in this system setting are only used to monitor the arrival events without any interference on their releases. The priority controller is only responsible for reassigning the priorities. All events are scheduled based on their assigned priorities. We assume, for every priorities reassignment, there is an instant interrupt to force the processor to process events based on the new priorities order. As there are multiple streams of low-critical events and all of them share one priority, they are served based on the principle of first-come-first-serve.

5.2. Workload-Shaping Policy

In contrast to the workload management by changing the execution priorities, the workload-shaping policy manages the low-critical workload by using a shaper to regulate the inflow of low-critical events. The priority of low-critical tasks is constantly set as the highest.

5.2.1. The Release of an Event. To ensure that the released event should keep the schedulability of all high-critical tasks, the WCET of a released event should not be larger than the longest feasible interference interval (LFII).

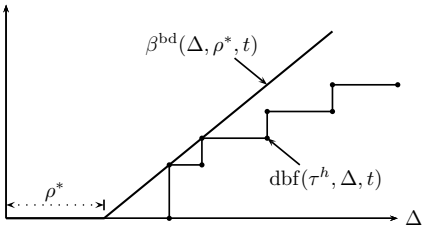


Fig. 8. An illustration for the LFII

Definition 5.3 (*Longest Feasible Interference Interval*). *The longest feasible interference interval* $\rho^*(t)$ *with respect to a given demand bound function* $\mathrm{dbf}(\tau^h, \Delta, t)$ *is defined as:*

$$\rho^*(t) = \max\{\rho^* : \beta^{\mathrm{bd}}(\Delta, \rho^*, t) \geq \mathrm{dbf}(\tau^h, \Delta, t),\ \forall \Delta \geq 0\}. \tag{20}$$

where $\beta^{\mathrm{bd}}(\Delta, \rho^*, t)$ *is a bounded-delay service curve that* $\beta^{\mathrm{bd}}(\Delta, \rho^*, t) = \max\{0, (\Delta - \rho^*)\},\ \forall \Delta \geq 0$.

The LFII is illustrated in Fig. 8, where $\mathrm{dbf}(\tau^h, \Delta, t)$ can be computed by using the backward derivation. $\beta^{\mathrm{bd}}(\Delta, \rho^*, t)$ means that the processing service is delayed for ρ^*. In this case, if the WCET of released event is smaller than the LFII, all high-critical tasks can still be schedulable as the provided service for them is larger than $\beta^{\mathrm{bd}}(\Delta, \rho^*, t)$. Therefore, during the runtime, by constantly computing the LFII, the

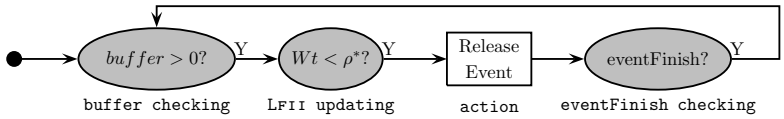


Fig. 9. The flow of workload-shaping policy

shaper decides how to release the arrival low-critical events. For every update of the LFII, the backward derivation has to be applied in deriving the $\mathrm{dbf}(\tau^h, \Delta, t)$, then the binary searching is used to search the maximum feasible LFII.

5.2.2. The Adaptive Shaping Flow. The flow of workload-shaping policy is seen in Fig. 9. The shaper has three states and one action, which are states of `buffer checking`, `LFII updating`, `eventFinish checking`, and the action of `release event`.

At the beginning, the shaper stays in `buffer checking`, in which the shaper constantly checks the emergence of an event in the buffer. Whenever there is an event in the buffer, the shaper transits to the `LFII updating`, in which the shaper updates the LFII, and compares it with the WCET of the event that is to be released. Once the LFII is greater than the WCET, this event is released, and the shaper transits to the `eventFinish checking`. The shaper stays in this state until the released event is finished, then the shaper transits back to the `buffer checking`.

When the shaper is in the `LFII updating`, the time for updating the LFII is based on the execution of high-critical events. The shaper is designed to update the LFII at the time when a high-critical event is finished. If the updated LFII is still less than the WCET of the event to be released, the LFII will be updated again when the next high-critical event is finished. The time setting for updating LFII is based on the fact that the demand of high-critical tasks will decrease if any high-critical event is finished, thus making the LFII greater.

6. A LIGHTWEIGHT METHOD

Both of schedulability analyzing approaches in Section 4 rely on the heavy computation, which prohibits their applications in the online cases. The computational overhead originates from three parts, i.e., RT computation, backward derivation, and the binary search for the LFII. To eliminate the heavy computation, a lightweight method is proposed in this section.

6.1. The Scenario of Setting the Low-Critical Priority as the Highest

Our mixed-criticality settings include a set of low-critical tasks and a set of high-critical tasks. To simplify the problem, we discuss how to compute a bound of low-critical workload in the scenario where the low-critical priority is higher than the priorities of all high-critical tasks. Suppose such a bound is ρ^*. A feasible ρ^* should guarantee that all high-critical tasks can be scheduled. Hence, the problem of obtaining the maximum ρ^* is that,

$$\textbf{maximize} \ \ \rho^*(t),$$

$$\textbf{s.t.} \ \ \beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t), \ \forall\, i \in [1..n], \tag{21}$$

$$\textbf{or s.t.} \ \ \beta^l(\tau^h, \Delta, t) \geq \mathrm{dbf}(\tau^h, \Delta, t). \tag{22}$$

This is a maximization problem with the constraint Eq. 21 or the constraint Eq. 22. The constraint Eq. 21 relies on solving n inequalities with forward RT computations, and the constraint Eq. 22 relies on backward deriving the group demand bound function. Both methods need to compute the complex (de-)convolution many times. To eliminate such complex computations, a leaky bucket [Le Boudec and Thiran 2001] is proposed to represent the workload arrival curve and a closed-form equation is derived for representing the provided service. In the following, we introduce the closed-form equation by analyzing a system with only two high-critical tasks.

6.1.1. Case for a System with Only Two High-Critical Tasks. Given two high-critical tasks τ_1^h and τ_2^h in a uniprocessor. Supposing at time t, their workload arrival curves are $\alpha^u(\tau_1^h, \Delta, t)$ and $\alpha^u(\tau_2^h, \Delta, t)$, and their demand bound functions are $\mathrm{dbf}(\tau_1^h, \Delta, t)$ and $\mathrm{dbf}(\tau_2^h, \Delta, t)$. Suppose the provided service is a bounded delay function $\beta^{\mathrm{bd}}(\Delta, \rho^*, t) = \max\{0, \Delta - \rho^*\}$. As shown in Fig. 10, to meet the deadlines of the tasks τ_1^h and τ_2^h, the following inequalities should hold:

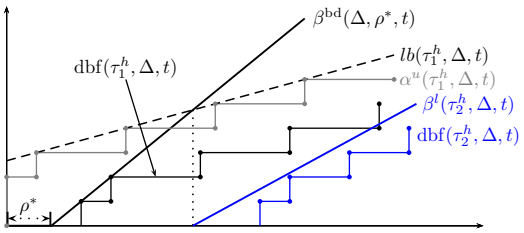


Fig. 10. The scheme for illustrating the schedulability analysis of two tasks

$$\beta^l(\tau_1^h, \Delta, t) \geq \mathrm{dbf}(\tau_1^h, \Delta, t), \quad \beta^l(\tau_2^h, \Delta, t) \geq \mathrm{dbf}(\tau_2^h, \Delta, t).$$

where

$$\beta^l(\tau_1^h, \Delta, t) = \beta^{\mathrm{bd}}(\Delta, \rho^*, t), \quad \beta^l(\tau_2^h, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau_1^h, \Delta, t), \alpha^u(\tau_1^h, \Delta, t)\Big).$$

If a leaky bucket $\mathrm{lb}(\tau_1^h, \Delta, t) = b_1(t) + r_1(t) \cdot \Delta$ is used to represent its workload arrival curve $\alpha^u(\tau_1^h, \Delta, t)$, then,

$$\beta^l(\tau_2^h, \Delta, t) = \mathrm{RT}\Big(\beta^l(\tau_1^h, \Delta, t), \mathrm{lb}(\tau_1^h, \Delta, t)\Big) = \sup_{0 \leq \lambda \leq \Delta} \Big\{ \max\{0, \lambda - \rho^*\} - b_1(t) - r_1(t) \cdot \lambda \Big\}.$$

As $\beta^l(\tau_2^h, \Delta, t) \geq 0$, we abbreviate $\beta^l(\tau_2^h, \Delta, t)$ as follows

$$\beta^l(\tau_2^h, \Delta, t) = \max\big\{0, (1 - r_1(t)) \cdot \Delta - \rho^* - b_1(t)\big\}.$$

Then, to keep the schedulability of both tasks, one only needs to guarantee the following two inequalities be true,

$$\max\{0, \Delta - \rho^*\} \geq \mathrm{dbf}(\tau_1^h, \Delta, t), \quad \max\big\{0, (1 - r_1(t)) \cdot \Delta - \rho^* - b_1(t)\big\} \geq \mathrm{dbf}(\tau_2^h, \Delta, t).$$

In this case, by using the leaky bucket to represent the original workload arrival curve, both $\beta^l(\tau_1^h, \Delta, t)$ and $\beta^l(\tau_2^h, \Delta, t)$ are derived to be rate-latency functions [Le Boudec and Thiran 2001]. The rates w.r.t. τ_1^h and τ_2^h are 1 and $1 - r_1(t)$, and the latencies w.r.t. τ_1^h and τ_2^h are ρ^* and $(\rho^* + b_1(t))/(1 - r_1(t))$. Actually, for a system with more than two high-critical tasks, if all workload arrival curves are represented by leaky buckets, the provided service for every high-critical task can be derived to be a closed-form equation. This closed-form euqation is also a rate-latency function.

6.1.2. Closed-Form Equation for the Provided Service. Analogous to the schedulability analysis with only two high-critical tasks, a similar procedure can be taken for analyzing a system with more than two high-critical tasks. As analyzed in Section 4, the following inequality sufficiently guarantees that all high-critical tasks can meet their deadlines.

$$\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t), \ \forall\, i \in [1..n].$$

To get $\beta^l(\tau_i^h, \Delta, t)$, a step-by-step forward RT computation should be used, which would block the computation speed. To remove this step-by-step computation, a closed-form equation is derived to represent the $\beta^l(\tau_i^h, \Delta, t)$.

THEOREM 6.1. *In a system with n high-critical tasks, the demand bound function $\mathrm{dbf}(\tau_i^h, \Delta, t)$ and workload arrival curve $\alpha^u(\tau_i^h, \Delta, t)$ are known at runtime. If a leaky bucket with the form of $\mathrm{lb}(\tau_i^h, \Delta, t) = r_i(t) \cdot \Delta + b_i(t)$ is used to conservatively represent $\alpha^u(\tau_i^h, \Delta, t)$, i.e., $\mathrm{lb}(\tau_i^h, \Delta, t) \geq \alpha^u(\tau_i^h, \Delta, t)$, the provided service $\beta^l(\tau_i^h, \Delta, t)$ for each task is as follows:*

$$\beta^l(\tau_i^h, \Delta, t) = \max\{0, (1 - R_i(t)) \cdot \Delta - \rho^* - B_i(t)\}, \tag{23}$$

where $R_i(t) = \sum_{j=1}^{i-1} r_j(t)$, $B_i(t) = \sum_{j=1}^{i-1} b_j(t)$, *and* $r_0(t) = 0$, $b_0(t) = 0$ *for brevity.*

PROOF. We prove this by induction.

If $n = 1$,

$$\beta^l(\tau_1^h, \Delta, t) = \max\{0, (1 - R_1(t)) \cdot \Delta - \rho^* - B_1(t)\} = \max\{0, \Delta - \rho^*\} = \beta^{\mathrm{bd}}(\Delta, \rho^*),$$

which is true.

We assume that Eq. 23 is true for the task $\tau_n^h (n \neq 1)$. Then, for the task τ_{n+1}^h, by using the Real-Time Interface analysis,

$$\beta^l(\tau_{n+1}^h, \Delta, t) = \sup_{0 \leq \lambda \leq \Delta} \Big\{ \max\{0, (1 - R_n(t)) \cdot \Delta - \rho^* - B_n(t)\} - b_n(t) - r_n(t) \cdot \lambda \Big\}.$$

As $\beta^l(\tau_{n+1}^h, \Delta, t) \geq 0$, $\beta^l(\tau_{n+1}^h, \Delta, t)$ can be rewritten as follows for brevity,

$$\beta^l(\tau_{n+1}^h, \Delta, t) = \max\left\{0, (1 - R_{n+1}(t)) \cdot \Delta - \rho^* - B_{n+1}(t)\right\}$$

□

COROLLARY 6.2. *By using the conservative representation* $\mathrm{lb}(\tau_i^h, \Delta, t) \geq \alpha^u(\tau_i^h, \Delta, t)$ *to compute* $\beta^l(\tau_i^h, \Delta, t)$*, under the constraint of Eq. 21, all high-critical tasks can meet their deadlines.*

PROOF. Let $\beta^{ACT}(\tau_i^h, \Delta, t)$ denote the actual provided service for task τ_i^h. As $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ is true for all high-critical tasks, one only needs to prove that the actual service $\beta^{ACT}(\tau_i^h, \Delta, t)$ is equal to or greater than $\beta^l(\tau_i^h, \Delta, t)$. We prove this also by induction.

When $n = 1$,

$$\beta^{ACT}(\tau_1^h, \Delta, t) = \beta^{\mathrm{bd}}(\tau_1^h, \Delta, t) = \beta^l(\tau_1^h, \Delta, t).$$

We assume $\beta^{ACT}(\tau_i^h, \Delta, t) \geq \beta^l(\tau_i^h, \Delta, t)$ is true for the task $\tau_n^h (n \neq 1)$. Then, for the task τ_{n+1}^h, we have

$$\beta^{ACT}(\tau_{n+1}^h, \Delta, t) = \mathrm{RT}\Big(\beta^{ACT}(\tau_n^h, \Delta, t), \alpha^u(\tau_n^h, \Delta, t)\Big) = \sup_{0 \leq \lambda \leq \Delta} \Big\{\beta^{ACT}(\tau_n^h, \lambda, t) - \alpha^u(\tau_n^h, \lambda, t)\Big\}.$$

As $\beta^{ACT}(\tau_n^h, \lambda, t) \geq \beta^l(\tau_n^h, \lambda, t)$ and $\mathrm{lb}(\tau_n^h, \lambda, t) \geq \alpha^u(\tau_n^h, \lambda, t)$, we have

$$\beta^{ACT}(\tau_{n+1}^h, \Delta, t) \geq \sup_{0 \leq \lambda \leq \Delta} \Big\{\beta^l(\tau_n^h, \lambda, t) - \mathrm{lb}(\tau_n^h, \lambda, t)\Big\} = \beta^l(\tau_{n+1}^h, \Delta, t).$$

Therefore, as $\beta^{ACT}(\tau_i^h, \Delta, t) \geq \beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ is true for all tasks, all deadlines can be met. □

6.1.3. Leaky Bucket Representation. From the Eqs. 8,5,6, we know the composition of $\alpha^u(\tau_i^h, \Delta, t)$ is the minimum of a set of staircase functions, and every staircase function is tracked by a dynamic counter. In principle, any leaky bucket can be used as long as this leaky bucket is equal to or greater than $\alpha^u(\tau_i^h, \Delta, t)$. But, in order to make our computation more tight, the leaky bucket should be as close to $\alpha^u(\tau_i^h, \Delta, t)$ as possible. Since the leaky bucket corresponding to the staircase function with the largest stair length in $\mathcal{U}(\tau_i^h, \Delta, t)$ is close to $\alpha^u(\tau_i^h, \Delta, t)$ in the long term, the staircase function with the largest stair length is thus used to compose a leaky bucket to represent the workload arrival curve.

Since the largest stair length in a workload arrival curve and the WCET of each task is known offline and unchanged during the runtime. Therefore, the leaky rate $r_i(t)$ is fixed to be $c_i/\delta_i^{\#}$, where $\delta_i^{\#}$ is the largest stair length. Hence, $1 - R_i(t)$ in Eq. 23 is also

known offline and fixed during the runtime. Then, to get $\beta^l(\tau_i^h, \Delta, t)$, one only needs to know the bucket size $b_i(t)$. Suppose for the task τ_i^h, $DC_i^\sharp$ is the counter for tracking the chosen staircase function in Algo. 1. At time t, it can be derived from Eqs. 8, 5, 6 that

$$b_i(t) = C(\tau_i, t) + c_i \cdot |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})| + c_i \cdot \begin{cases} DC_i^\sharp + \frac{t - k_i^\sharp \cdot \delta_i^\#}{\delta_i^\#} & \text{if } DC_i^\sharp < N_i^{\sharp u} \\ DC_i^\sharp & \text{if } DC_i^\sharp = N_i^{\sharp u} \end{cases} \tag{24}$$

where $k_i^\sharp$ is the auxiliary variable corresponding with $DC_i^\sharp$ in Algo. 1. Then, $b_i(t)$ is easy to get by just applying Eq. 24. $\beta^l(\tau_i^h, \Delta, t)$ can also be conveniently obtained as $R_i(t)$ is fixed and $B_i(t)$ is easy to obtain with the support of Eq. 24.

6.1.4. Computing $\rho^*(t)$. To solve the maximization problem with the constraint of Eq. 21, the first step is to use current parameters in monitors to update $\beta^l(\tau_i^h, \Delta, t)$ and $\mathrm{dbf}(\tau_i^h, \Delta, t)$. In this comparison, only a limited segment of Δ needs to be compared. If $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ in this limited segment, $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ in any interval Δ.

In a schedulable system, suppose for a high-critical task τ_i^h, at time t, there are $|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$ events that are backlogged in the buffer, and the absolute deadline trace is $D_{i,j}$, where j indicates the j-th event, as shown in Fig. 11. c_i is the WCET for this high-critical task τ_i^h.

LEMMA 6.3. *If $D_{i,x+1} - D_{i,x} = \delta_i^{max}$, where δ_i^{max} is the maximum stair length and $x > |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$, then for the absolute deadline of k-th event ($k \geq x$), we have*

$$D_{i,k+1} - D_{i,k} = \delta_i^{max}. \tag{25}$$

PROOF. From Eqs. 8, 6, we know the deadline trace is decided by $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ and $B(\tau_i^h, \Delta, t)$. As $x > |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$, the absolute deadline for the x-th event only depends on $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$. $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ is convex and is the minimum over all staircase functions. For the x-th and $(x+1)$-th events, if $D_{i,x+1} - D_{i,x} = \delta_i^{max}$, it indicates that $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ only depends on the staircase function with the largest stair length. For the k-th event ($k \geq x$), $\min_{j=1..n'}(\mathcal{U}_j(\tau_i^h, \Delta, t))$ also depends on the staircase function with the largest stair length. Hence, $D_{i,k+1} - D_{i,k} = \delta_i^{max}$. □

THEOREM 6.4. *Suppose the task τ_i^h is schedulable with a rate-latency function $\beta^l(\tau_i^h, \Delta, t)$. For the x-th and k-th event in lemma 6.3, if $\beta^l(\tau_i^h, D_{i,x}, t) \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t) \geq 0$, we have $\beta^l(\tau_i^h, D_{i,k}, t) \geq \mathrm{dbf}(\tau_i^h, D_{i,k}, t)$.*

PROOF. Assume $\beta^l(\tau_i^h, \Delta, t) = \max\{0, r \cdot \Delta + b\}$, as $\beta^l(\tau_i^h, D_{i,x}, t) \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t) \geq 0$, that is,

$$r \cdot D_{i,x} + b \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t),$$
$$r \cdot D_{i,x} + b + (k-x) \cdot c_i \geq \mathrm{dbf}(\tau_i^h, D_{i,x}, t) + (k-x) \cdot c_i.$$

From lemma 6.3, as $D_{i,k+1} - D_{i,k} = \delta_i^{max}$, $\mathrm{dbf}(\tau_i^h, D_{i,k+1}, t) - \mathrm{dbf}(\tau_i^h, D_{i,k}, t) = c_i$. We have

$$\mathrm{dbf}(\tau_i^h, D_{i,k}, t) = \mathrm{dbf}(\tau_i^h, D_{i,x}, t) + (k-x) \cdot c_i.$$

As $r > \frac{c_i}{\delta_i^{max}}$ for a schedulable system, we have

$$r \cdot D_{i,x} + b + (k-x) \cdot c_i \leq r \cdot D_{i,x} + b + r \cdot (k-x) \cdot \delta_i^{max} \leq r \cdot D_{i,k} + b$$

Therefore, $\beta^l(\tau_i^h, D_{i,k}, t) = r \cdot D_{i,k} + b \geq \mathrm{dbf}(\tau_i^h, D_{i,k}, t)$. □

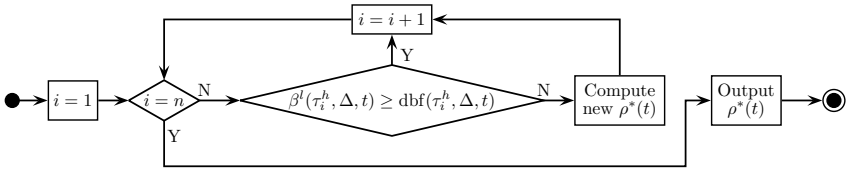


Fig. 12. The program diagram to compute the maximum $\rho^*(t)$ with the constraint of n inequalities

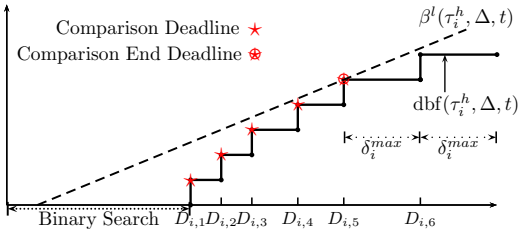


Fig. 11. An illustration how to do the comparison

Theorem 6.4 indicates that, if a rate-latency function $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t)$ within an interval of $[0, D_{i,x}]$, $\beta^l(\tau_i^h, \Delta, t)$ will be greater than $\mathrm{dbf}(\tau_i^h, \Delta, t)$ in any interval. In this paper, we define the earliest deadline that satisfies Eq. 25 as the comparison end deadline, as shown in Fig. 11. With the theorem 6.4, to do the comparison of Eq. 21, one only needs to compare $\beta^l(\tau_i^h, \Delta, t)$ with $\mathrm{dbf}(\tau_i^h, \Delta, t)$ before the comparison end deadline. For example, as shown in Fig. 11, there are only 5 deadlines before the comparison end deadline. If $\beta^l(\tau_i^h, D_{i,j}, t)$ is greater than $\mathrm{dbf}(\tau_i^h, D_{i,j}, t)$ in these 5 deadlines, this rate-latency function in other deadlines is also greater than the $\mathrm{dbf}(\tau_i^h, D_{i,j}, t)$. As $0 \leq \rho^*(t) \leq D_{i,1}$, we use the binary search to get the maximum $\rho^*(t)$. The computing complexity for one high-critical task is $O(log(n))$.

In short, for our proposed lightweight scheme, n inequalities need to be solved. As we only need to get the maximum $\rho^*(t)$ that makes all inequalities hold, it is not necessary to compute $\rho^*(t)$ for every inequality. The bubble sorting with one iteration can be used to pick out the maximum $\rho^*(t)$. It works like this, as shown in Fig. 12, we start the searching with a very large $\rho^*(t)$. As this large $\rho^*(t)$ will make $\beta^l(\tau_1^h, \Delta, t) < \mathrm{dbf}(\tau_1^h, \Delta, t)$, a new $\rho^*(t)$ will be computed by solving $\beta^l(\tau_1^h, \Delta, t) \geq \mathrm{dbf}(\tau_1^h, \Delta, t)$. This $\rho^*(t)$ is used in the second inequality. If the rate-latency function $\beta^l(\tau_2^h, \Delta, t)$ with $\rho^*(t)$ is greater than $\mathrm{dbf}(\tau_2^h, \Delta, t)$, this $\rho^*(t)$ is used in the third inequality. If not, we compute a new $\rho^*(t)$ of the second inequality, and use this new $\rho^*(t)$ in the third inequality. $\rho^*(t)$ is computed in this way until the last inequality. The computed $\rho^*(t)$ is the maximum LFII that satisfies all inequalities. The whole computing complexity is $O(n \cdot log(n))$.

6.2. The Lightweight Method in Workload Management Policies

We have introduced a lightweight method to compute the low-critical workload bound in the scenario where the priority of low-critical tasks is set as the highest. In this section, we analyze how to apply such lightweight method to our two proposed workload management policies.

6.2.1. The Lightweight Method in the Priority-Adjustment Policy. The problem of priority-adjustment policy is how to search a feasible priority for low-critical tasks, that is, to verify the system schedulability for every possible priorities assignment. Suppose the priorities assignment is that $P(\tau_1^h) > ... > P(\tau_k^h) > P(\tau^l) > P(\tau_{k+1}^h) > ... > P(\tau_n^h)$. Under this setting, we present how the proposed lightweight method can be used to verify the system schedulability.

Based on this priorities assignment, it can be derived that, by using the leaky bucket to represent the workload arrival curve for every task, the provided services for high-

critical tasks are that,

$$\beta^l(\tau_i^h, \Delta, t) = \begin{cases} \max\{0, (1 - R_i(t)) \cdot \Delta - B_i(t)\} & \text{if } i \leq k \\ \max\{0, (1 - R_i(t)) \cdot \Delta - B_i(t) - W(t)\} & \text{if } i > k \end{cases}, \tag{26}$$

where $R_i(t)$, $B_i(t)$ are the same as Eq. 23, and $W(t)$ is the workload of low-critical tasks at time t. Except the priorities setting, the derivation of Eq. 26 is the same as Eq. 23. For tasks τ_i^h where $i \leq k$, there is no interference from low-critical tasks, thus $\rho^* = 0$. For tasks τ_i^h where $i > k$, there is the interference of $W(t)$ from low-critical tasks. Such interference $W(t)$ can be considered as a leaky bucket $\mathrm{lb}(\tau^l, \Delta, t) = W(t) + 0 \cdot \Delta$. Therefore, the closed-form equation is still applicable.

With $\beta^l(\tau_i^h, \Delta, t)$ of Eq. 26, if $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t),\ \forall\ i \in [1..n]$ does not hold, the high-critical tasks are not schedulable, and the low-critical priority should be decreased, until $\beta^l(\tau_i^h, \Delta, t) \geq \mathrm{dbf}(\tau_i^h, \Delta, t),\ \forall\ i \in [1..n]$ holds.

6.2.2. The Lightweight Method in the Workload-Shaping Policy. The priorities setting in the workload-shaping policy is the same as the scenario in Section 6.1. Since the maximum $\rho^*(t)$ can be computed with a low overhead in the scenario, such $\rho^*(t)$ is used in managing the low-critical workload by the shaper.

7. IMPLEMENTATION AND EVALUATION

In this section, we evaluate the two proposed adaptive workload management policies and compare their performances with the offline approaches. The simulator is implemented in MATLAB by applying MPA and RTC/S tools [Wandeler and Thiele 2006] on a simulation host with Intel i7-4770 processor and 16 GB RAM.

7.1. Evaluation Setup

We use the system shown in Fig. 2 for our experiments. The model contains a set of low-critical tasks and a set of high-critical tasks, where each task set contains 5 independent tasks.

The activation pattern of high-critical tasks is a pjd model whose event arrival curve is shown in Eq. 1. For any high-critical task τ_i^h, the period p_i is a random integer from $[100, 300]\, ms$. The jitter j_i is set to be equal to period. The distance d_i is set to be an random integer from $[0, p]\, ms$. The relative deadline D_i is set to be equal to the period p_i. Each task utilization $U(\tau_i^h)$ and a task set utilization $U(\tau^h)$ are defined as $U(\tau_i^h) = c_i/p_i,\ U(\tau^h) = \sum_{i=1}^{n} c_i/p_i$, where c_i is the task WCET and n is the task number in a task set. Task utilizations are generated using the UUnifast [Bini and Buttazzo 2005], giving an unbiased distribution of utilization values. The WCET is set based on the utilization and selected period, i.e., $c_i = p_i \cdot U(\tau_i^h)$. The priorities assignment among high-critical tasks follows the principle of ensuring no high-critical task misses in the case that no low-critical interference exists. The Audsley's algorithm [Audsley 2001] is applied to assign feasible priorities to all high-critical tasks. In the case that no feasible priorities assignment is found for a task set, this task set is dropped and a new task set is generated until a feasible priorities assignment is found. In the simulation, there are four types of high-critical task set. The first type is named Type 1 whose utilization is 0.2. Successively, the second type is Type 2 with utilization 0.3, the third type is Type 3 with utilization 0.4, and the fourth type is Type 4 with utilization 0.5.

The low-critical tasks are activated sporadically, while their mean inter-arrival rates were chosen such that together they impose an additional utilization. Their mean inter-arrival intervals are in the interval $[50, 100]$. Denote the $U(\tau^l)$ as the utilization of the low-critical task set. All low-critical events follow the principle of first-come-first-service, so there is no individual priority for the low-critical task. We don't set the deadlines for low-critical events, and don't drop any low-critical events at runtime. For

a uniprocessor system, the utilization cannot exceed 1. Since the utilizations of the four high-critical task sets are $0.2, 0.3, 0.4, 0.5$, the ranges of low-critical utilization $U(\tau^l)$ w.r.t. Type 1, Type 2, Type 3, Type 4 are set to be $[0.1, 0.8]$, $[0.1, 0.7]$, $[0.1, 0.6]$, $[0.1, 0.5]$, in order to constrain the utilization of all tasks within 1.

In this work, six workload management approaches are evaluated, as shown in the following:

— Poffline: By setting the low-critical priority as the lowest, there is no interference on high-critical tasks. In this approach, the priority controller or shaper is not necessary.
— Soffline: By setting the low-critical priority as the highest, the low-critical workload is shaped to comply with the bound that is computed offline.
— Pexact: Applying the priority-adjustment policy proposed in Section 5.1 to manage the low-critical workload. For searching the feasible priorities assignment in this policy, the exact RT and backward computation are applied.
— Sexact: By setting the low-critical priority as the highest, the shaping policy proposed in Section 5.2 is used to adaptively shape the low-critical workload so that no high-critical tasks will miss their deadlines. The shaping bound is updated by applying the exact backward computation.
— Plight: In contrast to the Pexact method by applying the exact RT and backward computation, the lightweight method proposed in Section 6 is used to do the priority adjustment in this approach.
— Slight: In contrast to the Sexact method that applies the exact backward computation, the lightweight method proposed in Section 6 is used to update the shaping bound in this approach.

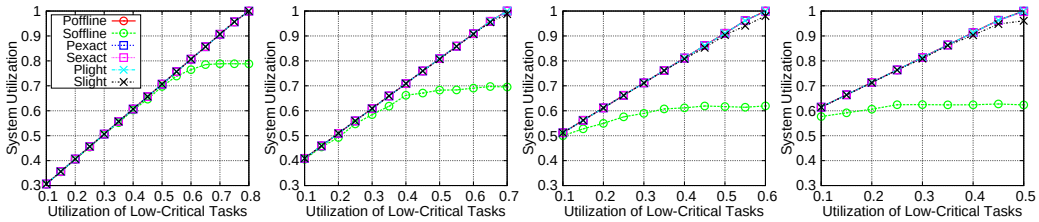
For every simulation, we simulate high-critical event traces with a 10 sec time span. In order to evaluate the performance of different workload management approaches, the following four metrics are used.

— System Utilization U_s: Suppose t_u is the cpu time within 10 sec that is used to process tasks. System utilization U_s is referred to $t_u/10$. U_s represents how much extent that the system processing capacity can be exploited.
— Average Response Time of Low-Critical Tasks R_L: This metric is referred to the average time span between the time that a low-critical event arrives and the time that this event is finished. R_L reflects the QoS of low-critical tasks.
— Latency Ratio of High-Critical Task Set L_H: Suppose D_i is the relative deadline of a high-critical task and R_i is its average response time, then the task set latency ratio is referred to be $\sum_{i=1}^{n} \Big(R_i/D_i\Big)/n$, where n is the task number in this task set. L_H reflects the influence of workload management approaches on the latency of high-critical tasks.
— Timing Overhead of Decision Making T_o: For both the exact and the lightweight methods, T_o is referred to the computation time needed to update the priority assignments or to release a low-critical event.

Note that U_s may not be equal to $U(\tau^h) + U(\tau^l)$ because LO-critical tasks may not be fully served.

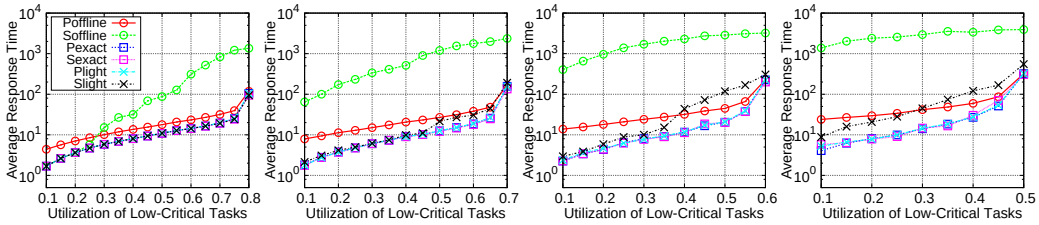
7.2. Simulation Results

In the following, we report simulation results for the listed four types, and the computation expenses by applying the exact computation and by applying the lightweight method in our proposed workload management policies. Under every specific setting of $U(\tau^h)$ and $U(\tau^l)$, 1000 test cases are generated to evaluate the performance of different workload management approaches. All the result figures are best seen in color online.



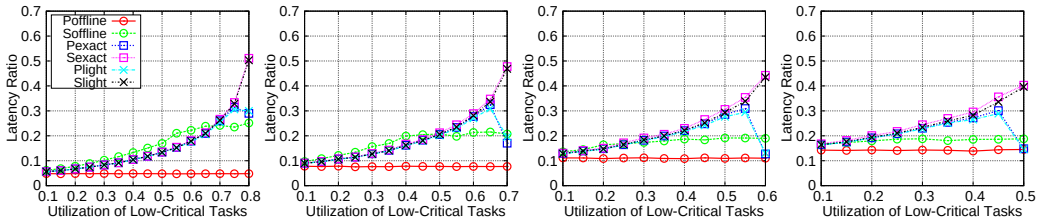
(a) Type 1 (b) Type 2 (c) Type 3 (d) Type 4

Fig. 13. The system utilization w.r.t the utilization of low-critical tasks



(a) Type 1 (b) Type 2 (c) Type 3 (d) Type 4

Fig. 14. The average response time of low-critical events



(a) Type 1 (b) Type 2 (c) Type 3 (d) Type 4

Fig. 15. The latency ratio of high-critical events

7.2.1. System Utilizations. The system utilizations of different workload management approaches working in the four types are shown in Fig. 13. From it, we find that, except Soffline, the system utilizations of using other approaches increase linearly with the utilization of low-critical tasks, and the highest system utilizations can almost reach 1. This shows that, all approaches except the Soffline, can fully make use of the system resources to process LO-critical and HI-critical tasks. The reason for the low system utilization of using Soffline is that the offline shaping bound is quite pessimistic, which results in a lot waste of processing resource.

7.2.2. Average Response Time of Low-Critical Tasks. The average response times of low-critical tasks w.r.t. four types of high-critical task sets are seen in Fig. 14. In general, we make three main observations.

— First, the shaping offline performs the worst among the six approaches. Fig. 14 shows that the average response time of using Soffline can be hundred times larger than that of using other methods. This is because the offline shaping bound is very pessimistic. If the utilization of low-critical events exceeds this bound, many low-critical events will be delayed for a long time.

— Second, the average response times of using Poffline are larger than those of using four online methods with the exception that Slight performs worse than Poffline when utilization of low-critical tasks is large in Types 3, 4. By setting the priority of all low-critical tasks as the lowest, low-critical events cannot be processed ahead of high-critical events, so the service that low-critical tasks receive is lower than that of applying the priority-adjustment policy. By using the online shaping, the low-

critical event can be processed ahead of the high-critical event only when the WCET of low-critical event is smaller than the computed LFII. In the four types, the WCET of low-critical event is smaller than the computed LFII in most cases, except the case of applying Slight when both $U(\tau^l)$ and $U(\tau^h)$ are greater than 0.3.

— Third, Sexact, Pexact, and Plight achieve almost the same average response time in the four types, while Slight performs badly in Type 3 and Type 4. This demonstrates that the lightweight computing method achieves the same results as using the exact computing methods by priority-adjustment policy. The lightweight method of workload-shaping policy is the same as the exact method only when $U(\tau^h)$ is small.

7.2.3. High-Critical Task Set Latency Ratio. The high-critical task set latency ratio w.r.t. four types of sets are seen in Fig. 15. In general, we make three main observations.

— Poffline has the lowest latency ratio, because the high-critical tasks are served without the interference from low-critical tasks.

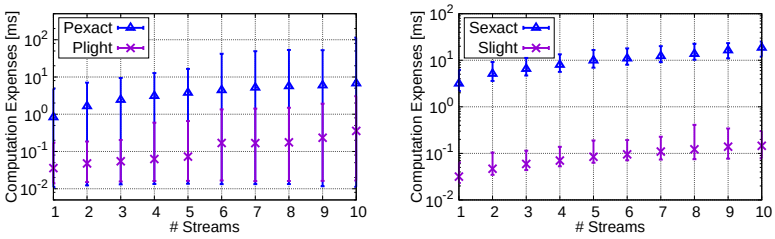
— In the four types, the latency ratio of Soffline first increases, then keeps constant. The latency ratio increases because the low-critical interference increases. However, the Soffline imposes a threshold on the low-critical interference. Once the low-critical interference exceeds this threshold, low-critical interference will be throttled, and the latency ratio of high-critical tasks will not increase. Besides, from Type 1 to Type 4 of Fig. 15, it can be seen that this threshold decreases with the increase of $U(\tau^h)$.

— With the increasing of $U(\tau^l)$, the latency ratios of Ponline and Plight first increase and then decrease. In the priority-adjustment policy, the latency ratio will be increased if high-critical tasks receive an increasing low-critical interference. However, as $U(\tau^l)$ increases, the low-critical workload also increases, which will result in that priorities of high-critical tasks will be set higher than low-critical priority after low-critical workload exceeds a certain threshold.

From Fig. 15, we also observe that the offline approaches have smaller latency ratio than the online approaches. This indicates that the online approaches sacrifice some QoS for high-critical tasks to improve the QoS of low-critical tasks. However, since the timing requirements of all high-critical tasks are sufficiently met, such sacrifice is worthwhile to improve the QoS of low-critical tasks.

7.2.4. Timing Overheads of Decision Making. For the priority-adjustment policy, the low-critical priority has to be adjusted when a low-critical event arrives or the execution of a low-critical event finishes, by using the Algo. 2 or Algo. 3 to decrease or increase the low-critical priority. We report the computation expenses of applying Algos. 2, 3 to adjust the priority. Fig. 16(a) shows the worst, best and average case computational expenses of using the exact and lightweight methods w.r.t. the number of high-critical event streams. In Fig. 16(a), the average computational expense of one stream is $0.83\,ms$ by using the exact method, and $0.034\,ms$ by using the lightweight method. From Algos. 2, 3, we know the complexity of searching feasible priority increases with the increase of the streams, which can be found that the computational expense increases with the increase of streams in this figure. In general, the computational expense of lightweight method is one order of magnitude lower than that of the exact method by priority-adjustment policy.

The workload-management policy depends on the LFII to shape the low-critical workload. The LFII is updated when the shaper is in `LFII updating state` and a high-critical event is finished. Fig. 16(b) shows the worst, best, and average case computational expenses of updating the LFII w.r.t. the number of high-critical event streams. The average computational expense of one stream is $3.15\,ms$ by using the exact method, and $0.031\,ms$ by using the lightweight method. For the ten streams, the average computational expenses of the two methods are $18.5\,ms$ and $0.14\,ms$. In



(a) Priority-adjustment policy (b) Workload-shaping policy

Fig. 16. Computation expense of the two adaptive workload management policies

general, the computational expense of lightweight method is two orders of magnitude lower than that of the exact method.

From the above results, we find that the workload-shaping policy is effective when low-critical events have low WCETs but may become ineffective when their WCETs are high. The workload-shaping policy is generally effective in regulating different kinds of events, while suffering the problem of frequent priority changes that will incur some extra runtime overheads. Therefore, from the perspective of implementations in a real platform, combining the two policies could be a possible solution that can overcome their own drawbacks and thus become more effective than using the two policies individually. In this article, since we focus on the evaluations of proof-of-concept simulations and the effectiveness of such practical implementations rely on the specific hardware platforms, the combination of the two approaches is thus not discussed.

8. CONCLUSIONS

In this article, we develop the adaptive workload management in MCSs to improve the QoS of low-critical tasks, while sufficiently guaranteeing the hard real-time constraints of high-critical tasks. The priority-adjustment policy and the workload-shaping policy have been presented. In order to make the two polices applicable in the online cases, the lightweight method was proposed to replace the complex RT computation and backward derivation in the schedulability verification during the runtime. Simulation results demonstrate the effectiveness of the two adaptive workload management policies, and show the low timing overhead of the lightweight method.

REFERENCES

Neil C Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.

Sanjoy K Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. 2011a. Mixed-criticality scheduling of sporadic task systems. In *Algorithms–ESA 2011*. Springer, 555–566.

Sanjoy K Baruah, Alan Burns, and Robert I Davis. 2011b. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS)*. 34–43.

Enrico Bini and Giorgio C Buttazzo. 2005. Measuring the performance of schedulability tests. *Real-Time Systems* 30, 1-2 (2005), 129–154.

Alan Burns and Sanjoy Baruah. 2013. Towards a more practical model for mixed criticality systems. *Real-Time Systems Symposium (RTSS)* (2013), 1–6.

Alan Burns and Robert Davis. 2015. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep* (2015).

Farhana Dewan and Nathan Fisher. 2012. Efficient admission control for enforcing arbitrary real-time demand-curve interfaces. In *Real-Time Systems Symposium (RTSS)*. 127–136.

François Dorin, Pascal Richard, Michaël Richard, and Joël Goossens. 2010. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems* 46, 3 (2010), 305–331.

Pontus Ekberg and Wang Yi. 2012. Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*. 135–144.

Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. 2016. Evaluation and Improvements of Runtime Monitoring Methods for Real-Time Event Streams. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 3 (2016), 56.

Biao Hu, Kai Huang, Gang Chen, and Alois Knoll. 2015. Evaluation of runtime monitoring methods for real-time event streams. In *Design Automation Conference (ASP-DAC)*. 582–587.

Biao Hu, Huang Kai, Gang Chen, Long Cheng, and Alois Knoll. 2015. Adaptive runtime shaping in mixed-criticality systems. In *international conference on Embedded software (EMSOFT)*.

Kai Huang, Gang Chen, Christian Buckl, and Alois Knoll. 2012. Conforming the runtime inputs for hard real-time embedded systems. In *Design Automation Conference (DAC)*. 430–436.

Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. 2011. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems* 47, 2 (2011), 163–193.

Kai Lampka, Kai Huang, and Jian-Jia Chen. 2011. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 267–276.

Kai Lampka, Simon Perathoner, and Lothar Thiele. 2009. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *Conference on Embedded Software (EMSOFT)*. 107–116.

Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer.

Moritz Neukirchner, Philip Axer, Tobias Michaels, and Rolf Ernst. 2013a. Monitoring of workload arrival functions for mixed-criticality systems. In *Real-Time Systems Symposium (RTSS)*. 88–96.

Moritz Neukirchner, Kai Lampka, Sophie Quinton, and Rolf Ernst. 2013b. Multi-mode monitoring for mixed-criticality real-time systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 34:1–34:10.

Moritz Neukirchner, Tobias Michaels, Philip Axer, Sophie Quinton, and Rolf Ernst. 2012. Monitoring arbitrary activation patterns in real-time systems. In *Real-Time Systems Symposium (RTSS)*. 293–302.

Linh TX Phan and Insup Lee. 2013. Improving schedulability of fixed-priority real-time systems using shapers. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 217–226.

Hang Su, Nan Guan, and Dakai Zhu. 2014. Service guarantee exploration for mixed-criticality systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*. IEEE, 1–10.

Hang Su and Dakai Zhu. 2013. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE, 147–152.

Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. 2000. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems (ISCAS)*. 101–104.

Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. 2006. Real-time interfaces for composing real-time systems. In *Conference on Embedded Software (EMSOFT)*. 34–43.

Sebastian Tobuschat, Moritz Neukirchner, Leonardo Ecco, and Rolf Ernst. 2014. Workload-aware shaping of shared resource accesses in mixed-criticality systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 35.

Steve Vestal. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium (RTSS)*. 239–243.

Ernesto Wandeler. 2006. *Modular performance analysis and interface-based design for embedded real-time systems*. Ph.D. Dissertation. ETH Zurich.

Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. 2012. On the use of greedy shapers in real-time embedded systems. *ACM Trans. Embed. Comput. Syst.* 11, 1 (2012), 1:1–1:22.

Ernesto Wandeler and Lothar Thiele. 2005. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Conference on Embedded Software (EMSOFT)*. 80–89.

Ernesto Wandeler and Lothar Thiele. 2006. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`. (2006). http://www.mpa.ethz.ch/Rtctoolbox