Adaptive Runtime Shaping for Mixed-Criticality Systems

Biao Hu[1],Kai Huang[1,2], Gang Chen[1], Long Cheng[1], Alois Knoll[1]
[1]Tech. Univ. Muenchen TUM, [2]Sun Yat-Sen University
[1]{hub,huangk,cheng,chengl,knoll}@in.tum.de, [2]huangk36@mail.sysu.edu.cn

ABSTRACT

This paper investigates runtime shaping for mixed-criticality systems to increase the system QoS. Unlike the previous work in the literature that enforces an offline workload bound, an adaptively shaping approach is proposed where the incoming workload of the low-critical tasks is regulated by the actual demand of the high-critical tasks. This actual demand is adaptively updated using the historical arrival information of the high-critical tasks and thus can maximize the runtime QoS of low-critical tasks. To reduce the online overheads of computing the workload demand, a lightweight scheme with the complexity of $O(n\log(m))$ is developed. Experiments are also provided to demonstrate the effectiveness and efficiency of our approach.

1. INTRODUCTION

Nowadays, the integration of components with different levels of criticality onto a shared hardware platform is preferable in order to increase the resource utilization and reduce the cost of hardware. In particular for the traditional safety-critical avionics and automotive domains, the system evolves into a mixed-criticality system by which the stringent non-functional requirements w.r.t. cost, space, weight, heat generation, and power consumption can be met [3]. For example, an unmanned aerial vehicle often integrates high-critical tasks, such as flight control and actuation control, with low-critical tasks (like the mission function), together into a same processor [9].

The schedule of a mixed-criticality system is usually priority-based, where the low-critical tasks are set with higher priorities in order to improve their QoS. To guarantee the execution of the high-critical tasks, the execution for the high-priority low-critical tasks are constrained to a certain bound [17, 19, 22]. When their execution demand exceeds this bound, either the executions are delayed by a regulator [22] or the execution priorities are exchanged with the high-critical tasks [19]. Nevertheless, computing such a bound for the low-critical tasks is non-trivial as the bound on the one hand should be high enough to guarantee the execution of the high-critical tasks and on the other hand should not be too pessimistic as to hamper the QoS of low-critical tasks.

There is related work in the literature to offline compute workload bound by which the incoming workload of the low-criticality are determined at runtime. Wandeler *et al.* [22,23] proposed to use the Real-Time Interface to obtain the shaping bound and suggested delaying the non-real-time events when they exceed the precomputed bound. With the computed shaping bound, together with the sensitivity analysis [16], Neukirchner *et al.* [13, 15] discussed in more details 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.* [19] presented a scheme to exploit throughput and latency slack of critical applications by prioritizing non-critical over critical accesses and exchanging the priorities when the non-critical workload exceeds the bounds.

The shaping 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 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 adaptation is however not easy. On the one hand, the actual demand should be a valid upper bound that can guarantee that no high-critical tasks miss their deadlines. On the other hand, the adaptation decisions should be lightweight. While most of the previous work relies on heavy numerical computation, to the best of our knowledge, no work so far in the literature has considered adaptively refining the shaping bound at runtime.

Being aware of this, this paper proposes an adaptive shaping scheme for mixed-criticality systems scheduled by the preemptive fixed-priority policy. The shaping bound for the low-critical tasks is dynamically refined during the runtime, using the historical knowledge of the workload arrival of the high-critical tasks in the past. To model non-deterministic workload arrivals, the arrival curve that models workload in time-interval domain has been adopted. To compute the history-aware future workload for the high-critical tasks, the so-called dynamic counters [10] are used. Real-Time Interface [23] is used to compute the maximum allowable interference from low-critical tasks. The detailed contributions are as follows:

- We present an adaptive scheme for shaping the low-critical workload online in mixed-criticality systems.
- A lightweight method with complexity of $O(m \cdot log(n))$ is proposed to refine the shaping bound. The timing overhead is two orders of magnitude less than the backward derivation method.
- Experimental results show the efficiency and effectiveness of our presented shaping scheme.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 introduces the system model and presents the background knowledge. Section 4 presents the problem of shaping by enforcing an offline bound. Section 5 presents our approaches and the shaping policy. Section 6 presents the experimental results. Section 7 concludes the paper.

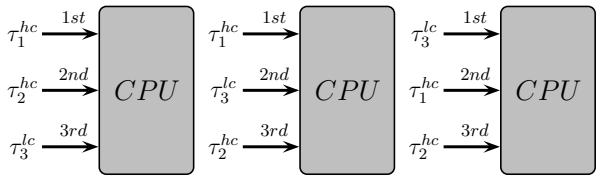
2. RELATED WORK

In this paper, we address the problem of providing an efficient mechanism to dynamically shape the incoming workload of low-critical tasks at runtime in mixed-criticality real-time systems. Our proposed method builds on three research aspects, which are the Real-Time Interface analysis, the workload prediction and the runtime shaping.

Real-Time Interface analysis. Real-Time Interface analysis provides the answer to these questions: does the composed system satisfy all requested real-time properties, such as delay and throughput constraints. The central idea behind interface-based design is to describe components by a component interface. The principles of interface-based design are described in [18]. The stateless Assume/Guarantee Real-Time Interfaces, proposed in [23], connect the theory of Real-Time Calculus into the interface-based design. By applying the (de-)convolution from Real-Time Calculus, the need for the classical binary search approach to find an economically dimensioned system is partly removed. This approach leads to a faster design process.

Workload Prediction. System workload prediction is based on the principle that the runtime workload cannot exceed the provided bounds. In Real-Time Calculus, a task activation is often modeled as an event. The arrival curves, originated from Network Calculus [12], provide an upper and lower bound on the number of arrival events in any time interval. In [7, 8], a prediction method is proposed based on a certain length of historical information of event arrival. This prediction is not tight as the historical events within only a certain length can be used. As arrival curves can be approximated by several staircase functions, a more tight prediction is proposed in [10] by using dynamic counters to track the runtime burst for every staircase function.

Runtime Shaping. The shapers are often used in regulating the packets in network [12]. The use of greedy shapers in scheduling the real-time system is analyzed in [22]. In [6], by implementing a dual-bucket mechanism (similar to dynamic counters) into FPGA, the runtime inputs are shaped to the designed arrival curves. The experiment shows the overhead of this shaping mechanism is very low in FPGA. Another lightweight monitoring scheme based on the minimum distance functions (i.e., an inverse representation of arrival curves) is proposed to do the event model verification [14]. Based on this method, the multi-mode monitoring is proposed in [15] to monitor the low-critical task workload by switching the workload bound distribution. A similar work is to monitor the low-critical task workload as a group by using the workload arrival function [13]. In [5], the two model verification methods based on the arrival curve and on the minimum distance function are evaluated. Phan *et al.* [17] introduced a technique to use an optimal greedy shaper for periodic tasks with jitter to improve the schedulability of real-time systems. This technique only targets at the periodic tasks with jitter.



(a) Non-shaping (b) Scheduling (c) Shaping

Figure 1: Three settings of priority assignment

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

3. MODELING AND BACKGROUND

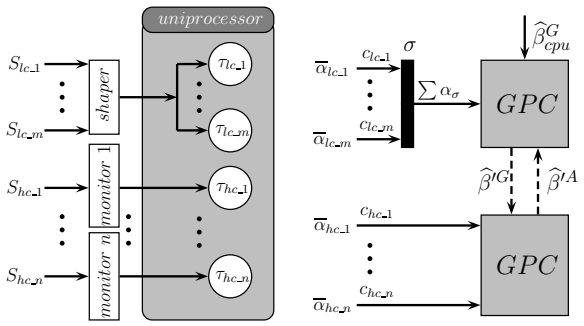
This section introduces the system model and background knowledge used in this paper.

3.1 Priority Assignment

The schedule of a mixed-criticality system is usually preemptively priority-based. There are three typical priority settings. As shown in Fig. 1, two high-critical tasks τ_1^{hc}, τ_2^{hc}, and one low-critical task τ_3^{lc} are running in a processor. In the first setting (Fig. 1(a)), in order to remove the interference on the high-critical tasks, the low-critical task τ_3^{lc} is set with the lowest priority. This setting makes the shaping on the low-critical workload not necessary because the low-critical task cannot interfere with the executions of high-critical tasks. This setting, however, is not helpful for improving the QoS of low-critical tasks. Recent research [1, 20] shows that the system is still schedulable even if the priorities of low-critical tasks are set higher than some high-critical tasks, as shown in Fig. 1(b). The assumption for this case is that all tasks are sporadically activated, and the minimum interval between two task activations are known. A survey of this case can be seen [3]. While the assumption of priori knowledge to the activation patterns of high-critical tasks is valid in most cases, the activations of low-critical tasks might not be sporadic. The low-critical tasks, such as entertainment applications in a car, are often not predictable [13, 15]. To improve the QoS of low-critical tasks, while guaranteeing the deadline requirements of high-critical tasks, another priority setting is proposed, as shown in Fig. 1(c). All low-critical tasks are set with higher priorities. In this setting, to guarantee the execution of high-critical tasks, the execution for high-priority low-critical tasks are constrained within a certain bound. This priority assignment is adopted in recent work, such as [13, 15].

3.2 System Models

We consider a uniprocessor system that is scheduled according to the preemptive FP scheduling policy. As shown in Fig. 2(a), a set of low-critical tasks $S_{lc} = \{\tau_{lc\_1}, ..., \tau_{lc\_m}\}$ and a set of high-critical tasks $S_{hc} = \{\tau_{hc\_1}, ..., \tau_{hc\_n}\}$ are running on this uniprocessor, and the priorities for the set of low-critical tasks are higher than the priorities for the set of high-critical tasks. Without loss of generality, the tasks $\tau_{lc\_1}, ..., \tau_{lc\_m}$ in S_{lc} and the tasks $\tau_{hc\_1}, ..., \tau_{hc\_n}$ in S_{hc} are ordered according to their priorities, i.e., the priority of the task $\tau_{h(l)c\_i}$ is higher than that of the task $\tau_{h(l)c\_j}$ when $i < j$.



(a) System model (b) System abstraction

Figure 2: A mixed-criticality system scheduled by the preemptive FP policy

A task is activated by an activating event, and is assumed to be executed for its WCET. The task activations can be expressed as an event stream. A trace of such an event stream can be conveniently 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 $\overline{\alpha}(\Delta) = [\overline{\alpha}^u(\Delta), \overline{\alpha}^l(\Delta)]$ of upper and lower arrival curves provides an abstract event stream model that characterizes a whole class of (non-deterministic) event streams. $\overline{\alpha}^u(\Delta)$ and $\overline{\alpha}^l(\Delta)$ provide an upper and lower bound on the number of events seen on an event stream in any time interval of length Δ [21]:

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

with $\overline{\alpha}^l(\Delta) = \overline{\alpha}^u(\Delta) = 0$ for $\Delta \leq 0$.

The concept of an arrival curve unifies many other common timing models of event streams. For example, for an event with period p, jitter j, and minimal inter arrival distance d, the upper arrival curve is $\overline{\alpha}^u(\Delta) = \min\{\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$.

Analogous to arrival curves that provide an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves then provides an abstract resource model, which shows an upper and lower bound on the available resources in any time interval of length Δ. As an arrival curve $\overline{\alpha}_i$ specifies the event and a service curve β specifies the available processing time, the arrival curve $\overline{\alpha}_i(\Delta)$ has to be transformed to the arrival curve α_i to indicate the amount of computation time required for the arrived events in intervals. Suppose that the WCET of an event stream is c_i. Then, the transformation can be done by $\alpha_i^u = c_i \overline{\alpha}_i^u$, $\alpha_i^l = c_i \overline{\alpha}_i^l$ and back by $\overline{\alpha}_i^u = \alpha_i^u / c_i$, $\overline{\alpha}_i^l = \alpha_i^l / c_i$.

For the high-critical tasks, their activation patterns are often fixed. Therefore, the lower and upper arrival curves for them can be decided in the system design-time stage. For the low-critical tasks, such as multi-media tasks in a car, their workload may sometimes be overloaded. Hence, their arrival events must be shaped. In our system, the monitors are set in every input port of high-critical tasks to track the actual events. To enforce the workload of low-critical tasks is not overloaded, a shaper is set to regulate the input events of all low-critical tasks.

3.3 Prediction with Dynamic Counters

In principle any (discrete) complex arrival pattern can be bounded by a set of upper and lower staircase functions of the form $\overline{\alpha}_i^u(\Delta) = N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor$ [11]:

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \overline{\alpha}^u(\Delta) \leq \min_{i=1..n} \{\overline{\alpha}_i^u(\Delta)\}.$$

Algorithm 1 Implement a dynamic counter to track a staircase function

Input: signal s, $\triangleright tuple < DC_i, CLK_i >$;

```
1: if s = eventArrival then
2:     if DC_i = N_i^u then
3:         reset_timer(CLK_i, δ_i^u)
4:         k_i = 0
5:     end if
6:     DC_i ← DC_i − 1
7: end if
8: if s = CLK_i Timeout then
9:     DC_i ← min(DC_i+1, N_i^u)
10:    reset_timer(CLK_i, δ_i^u)
11:    k_i = k_i + 1
12: end if
13: if DC_i < 0 then
14:    report_exception
15: end if
```

A dynamic counter (DC_i) can be used to track a single upper staircase function ($\overline{\alpha}_i^u$). The detail tracking algorithm can be seen in Algo. 1 [10]. DC_i tracks the potential burst capacity, and the auxiliary variable k_i in Algo. 1 tracks the offset between the current time t and the last δ_i^u. Below shows an example of using dynamic counters for event prediction.

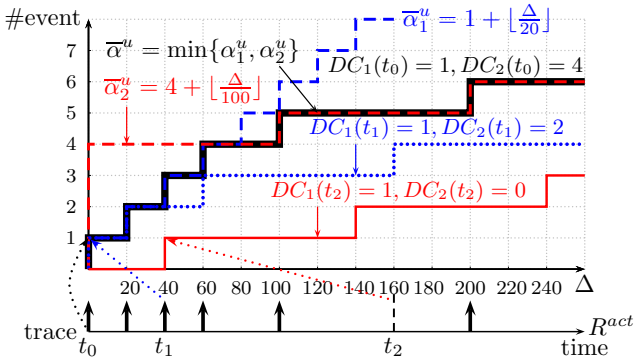


Figure 3: An example for using dynamic counters to predict the future events

EXAMPLE 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 [10], 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 $DC_i(t)$, together with staircase function, yields the following future events prediction [10]:

$$\mathcal{U}_i(\Delta, t) = DC_i(t) + \begin{cases} \lfloor \frac{\Delta + (t - k_i\delta_i^u)}{\delta_i^u} \rfloor & \text{if } DC_i(t) < N_i^u \\ \lfloor \frac{\Delta}{\delta_i^u} \rfloor & \text{if } DC_i(t) = N_i^u \end{cases} \tag{1}$$

where N_i^u is the absolute burst. The above function bounds future event arrivals. By using the monitor to track the events trace for high-critical tasks, DC_i and k_i can be known during the runtime. Therefore, $\mathcal{U}_i$ is also known during the runtime. For bounding the number of future event arrivals w.r.t. complex task activations, one can simply take the minimum over all the $\mathcal{U}_i$:

$$\mathcal{U}(\Delta, t) = \min_{i=1..n}(\mathcal{U}_i(\Delta, t)). \tag{2}$$

4. OFFLINE BOUND AND MOTIVATION

In this section, we derive the maximum offline service bound for the shaper to regulate the workload of all low-critical tasks. For event streams described by arrival curves, one can apply Real-Time Interface [23] to verify whether a system can provide enough resource guarantee service. We also present the problem of using the offline bound to shape low-critical workload.

4.1 Offline Bound Analysis

The system model is abstracted as the system abstraction shown in Fig. 2(b), where $\widehat{\beta}^G_{cpu}$ denotes the whole processor service, $\sum \alpha_\sigma$ denotes the arrival workload of low-critical tasks after shaping, $\widehat{\beta}'^G$ denotes the resource guarantee service after processing $\sum \alpha_\sigma$, and $\widehat{\beta}'^A$ denotes the service demand bound that is needed to meet the deadlines of all high-critical tasks. For a schedulable real-time system, the following equation has to be true:

$$\widehat{\beta}'^G(\Delta) \geq \widehat{\beta}'^A(\Delta), \qquad \forall \Delta \geq 0. \tag{3}$$

As our system is scheduled by the preemptive FP policy, the resource guarantee service is that:

$$\widehat{\beta}'^G = RT(\widehat{\beta}^G_{cpu}, \sum \alpha_\sigma)^1 \tag{4}$$

By inverting Eq. 4, we can get

$$\sum \alpha_\sigma = RT^{-\alpha}(\widehat{\beta}'^G, \widehat{\beta}^G_{cpu})^2 \tag{5}$$

As $\widehat{\beta}'^G \geq \widehat{\beta}'^A$ for a schedulable system, the maximum acceptable arrival workload of low-critical tasks is that:

$$\sum \alpha_\sigma = RT^{-\alpha}(\widehat{\beta}'^A, \widehat{\beta}^G_{cpu}). \tag{6}$$

To get $\sum \alpha_\sigma$, one has to know $\widehat{\beta}'^A$. Suppose there are multiple high-critical tasks, the total service demand bound $\widehat{\beta}'^A$ can be derived by backward derivation with the deadline service demand for every high-critical task.

[1] $RT(\beta, \alpha) = \sup_{0 \leq \lambda \leq \Delta} \{\beta(\lambda) - \alpha(\lambda)\}$

[2] $RT^{-\alpha}(\beta', \beta)(\Delta) = \beta(\Delta + \lambda) - \beta'(\Delta + \lambda)$ for $\lambda = \sup\{\tau : \beta'(\Delta + \tau) = \beta'(\Delta)\}$

THEOREM 1. *[8,23] Suppose a multiple tasks scheduled by FP policy in a system as shown in Fig. 4, where $\widehat{\beta}'^A_i$ denotes the service demand for meeting the relative deadlines of tasks from τ_{hc_n} to τ_{hc_i}, and $\widehat{\beta}'^A_1 = \widehat{\beta}'^A$. The following equation has to be true for a schedulable system,*

$$\widehat{\beta}'^A_i \geq \max\{\alpha_{hc\_i}(\Delta - D_i), RT^{-\beta}(\widehat{\beta}'^A_{i+1}, \alpha_{hc\_i})\}^3 \tag{7}$$

where D_i is referred as the relative deadline of task τ_{hc_i}, and $\widehat{\beta}'^A_{n+1}$ is defined as 0 for brevity.

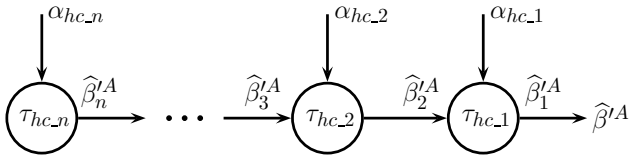


Figure 4: The flow of backward deriving the total service demand for FP scheduling

As a trace constrained by any wide-sense increasing arrival function is equivalent to this trace constrained by its corresponding subadditive closure [12], the shaping bound is ensured to be subadditive closure without any loss. To achieve this, the shaping bound $\sum \alpha_\sigma$ is computed by the following equation:

$$\sum \alpha_\sigma = \min\{\sum \alpha_\sigma, (\sum \alpha_\sigma \otimes \sum \alpha_\sigma), ...\}^4. \tag{8}$$

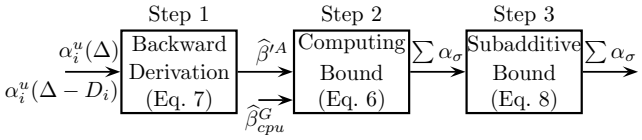


Figure 5: Compute the shaping bound

In short, to get the offline shaping bound, one can follow the computation steps shown in Fig. 5. The first step is to compute the service demand $\widehat{\beta}'^A$ by applying backward derivation with $\alpha^u_i(\Delta)$ and $\alpha^u_i(\Delta - D_i)$. The second step is to compute the bound by applying Eq. 6 with $\widehat{\beta}'^A$ and $\widehat{\beta}^G_{cpu}$. The last computation step is to make the shaping bound $\sum \alpha_\sigma$ subadditive.

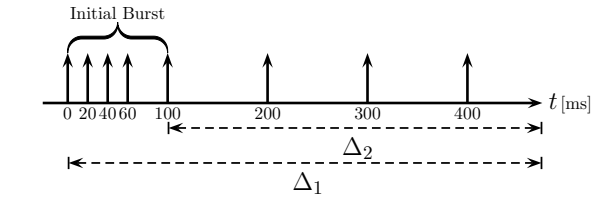
4.2 Motivation Example

The problem of shaping by $\sum \alpha_\sigma$ (Eq. 8) is that, the shaper is independent with the actual executions of high-critical tasks, thus the shaping bound sometimes is too pessimistic as the shaper assumes the execution of high-critical tasks are always worst cases. We use a simple example to illustrate this problem.

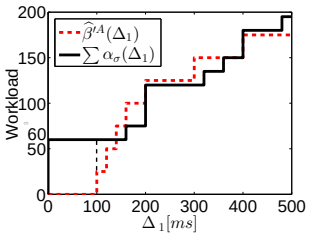
EXAMPLE 2. *In a mixed-criticality system, a low-critical task with a high priority and a high-critical task with a low priority are scheduled in a processor. The high-critical task is a PJD model, which is characterized with a period $p = 100\,ms$, a jitter $j = 300\,ms$, and a minimum distance $d = 20\,ms$. The* WCET *is $25\,ms$. As shown in Fig. 6(a), for the high-critical task, the worst-case activations pattern is*

[3] $RT^{-\beta}(\beta, \alpha)(\Delta) = \beta(\Delta - \lambda) + \alpha(\Delta - \lambda)$ for $\lambda = \sup\{\tau : \beta(\Delta - \tau) = \beta(\Delta)\}$

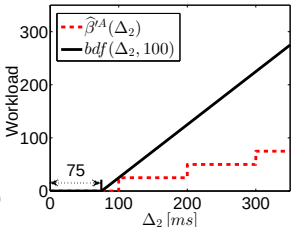
[4] $(f \otimes g)(\Delta) = \inf_{0 \leq \lambda \leq \Delta} \{f(\Delta - \lambda) + g(\lambda)\}$



(a) Event arrival time frame



(b) Initial time



(c) After the initial burst

Figure 6: Motivating example

an initial burst during the first 100 *ms, and after that, the task is activated every* 100 *ms.*

Following the steps of the previous section, the offline workload bound $\sum \alpha_\sigma$ *can be computed by applying the Eqs. 7, 6, 8. The service demand* $\widehat{\beta}'^A(\Delta_1)$ *and workload bound* $\sum \alpha_\sigma(\Delta_1)$ *(bold line) are shown in Fig. 6(b). It can be seen that within an interval of* 150 *ms, only* 60 *ms low-critical workload is allowed. Now, we consider another case. We assume the initial burst of the high-critical task happens within* 100 *ms, and there is no low-critical workload within this* 100 *ms. If dynamic counters are used to predict its future events, we know the actual demand of high-critical task at time* $t = 100\,ms$ *is bounded by* $\widehat{\beta}'^A(\Delta_2)$*, as shown in Fig. 6(c). Suppose a bounded delay function* $bdf(\Delta, t)$ *is used to serve this high-critical task, where* Δ *denotes the interval and* t *denotes the current time. There will be no deadline miss as long as* $bdf(\Delta_2, 100)$ *is greater than* $\widehat{\beta}'^A(\Delta_2)$*. The largest delay is* 75 *ms, which indicates that the service for high-critical task can be delayed for* 75 *ms. It also means that, during this delay, low-critical tasks can be processed. Therefore, the largest low-critical workload at* $t = 100\,ms$ *can be* 75 *ms, instead of bounded by* 60 *ms.*

As this example illustrates, if the shaper shapes the workload of low-critical task based on the offline bound, the QoS for low-critical tasks is not as high as the shaping based on the largest delay of bounded delay function at runtime. Besides, since the shaper relies on updating the largest delay to refine the shaping bound, instead of using the sub-additive bound, step 2 and step 3 are not necessary for online shaping. The approach using the largest delay to shape low-critical workload is similar to the energy management in [8], where the longest sleeping interval that the processor can have is updated to manage the processor state while guaranteeing no tasks miss their deadlines. In this paper, we propose to update the largest delay of bounded delay function based on the actual service demand to shape the low-critical workload.

5. OUR APPROACH

As illustrated in the previous example, computing an adaptive shaping bound at runtime is theoretically possible,

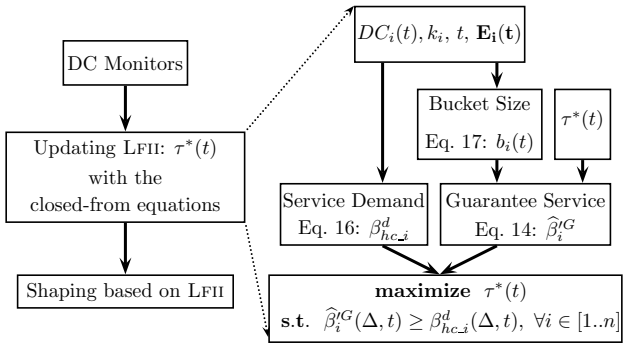


Figure 7: Main flow of our approach

i.e., using the dynamic counters to predict the future demand for individual streams and Eq. 7 to backward derive the total service demand with individual demands, and using binary search to compute the largest delay of bounded delay function. However, the computational overhead prohibits the application of such method for online uses. To eliminate the heavy computation, a lightweight approach is proposed in this section.

5.1 Adaptive Shaping Scheme Overview

Analogous to the largest delay of a bounded delay function, we formally define a shaping bound at runtime, i.e., the longest feasible interference interval (LFII), as follows.

DEFINITION 1. *(Longest Feasible Interference Interval) The longest feasible interference interval* $\tau^*(t)$ *with respect to a given service demand* $\widehat{\beta}'^A(\Delta, t)$ *is defined as:*

$$\tau^*(t) = \max\{\tau : bdf(\Delta, \tau) \geq \widehat{\beta}'^A(\Delta, t),\ \forall \Delta \geq 0\}. \qquad (9)$$

where $\widehat{\beta}'^A(\Delta, t)$ *is the service demand for meeting all high-critical tasks deadlines, and* $bdf(\Delta, \tau)$ *is a bounded delay function with* $bdf(\Delta, \tau) = \max\{0, (\Delta - \tau)\},\ \forall \Delta \geq 0$.

From this definition, we know that the $\tau^*(t)$ indicates the largest acceptable low-critical workload at time t. By constantly updating the $\tau^*(t)$, the shaping bound is adaptively refined. An approach to update $\tau^*(t)$ is to backward derive the $\widehat{\beta}'^A(\Delta, t)$ and compute the $\tau^*(t)$ by solving Eq. 9. This backward derivation is computational intensive, which is not suitable for online applications.

We consider this problem from another side. Based on the truth that all high-critical tasks can meet their deadlines if the resource guarantee service for every high-critical task is greater than its deadline service demand, one only needs to compute a $\tau^*(t)$ that satisfies this condition. Therefore, the computation of $\tau^*(t)$ is transformed to a maximization problem. The object is the maximum $\tau^*(t)$ with the constraint that the resource guarantee service $\widehat{\beta}'^G_i$ for every high-critical task is greater than its deadline service demand $\beta^d_{hc\_i}$. That is,

$$\textbf{maximize}\ \ \tau^*(t),$$
$$\textbf{s.t.}\ \ \widehat{\beta}'^G_i(\Delta, t) \geq \beta^d_{hc\_i}(\Delta, t),\ \forall i \in [1..n].$$

The main flow of our approach is shown in Fig. 7. We use dynamic counters to constantly monitor the high-critical events. As the workload arrival curve α_{hc_i} and deadline

service demand $\beta^d_{hc\_i}$ for every high-critical task are closed-form equations w.r.t. runtime parameters in the monitor, α_{hc_i} and $\beta^d_{hc\_i}$ are known at runtime. By using a leaky bucket to represent the workload arrival curve, we derive a closed form for the resource guarantee service $\widehat{\beta}'^G_i$. This closed-form equation is a rate-latency function [12] whose rate can be fixed offline and the latency is composed of $\tau^*(t)$ and the bucket size. The bucket size is a closed-form equation w.r.t. runtime parameters in the monitors. Therefore, by assuming a $\tau^*(t)$, $\widehat{\beta}'^G_i$ is also known at runtime. Since $\widehat{\beta}'^G_i$ is a linear function, computing $\tau^*(t)$ is a linear programming, which has a very low overhead.

5.2 Shaping Bound Update with Closed Form

In this section, we present the solution of the maximization problem. We first use an example with two high-critical tasks to illustrate our approach, then this approach is extended to arbitrary numbers of high-critical tasks. Further, we derive how to use the parameters in the monitors to get the bucket size and the deadline service demand. Lastly, we show how to compute the maximum $\tau^*(t)$ with the constraint of n inequalities.

5.2.1 Case for two high-critical tasks

Given two high-critical tasks τ_{hc_1} and τ_{hc_2} in a mixed-criticality system. Suppose at time t, their workload arrival curves are $\alpha_{hc\_1}(\Delta,t)$, $\alpha_{hc\_2}(\Delta,t)$, and their deadline service demands are $\beta^d_{hc\_1}(\Delta,t)$ and $\beta^d_{hc\_2}(\Delta,t)$. Suppose the LFII is $\tau^*(t)$ at time t, as shown in Fig. 8. To meet the deadlines of the task τ_{hc_1}, τ_{hc_2}, the following inequalities should hold.

$$\widehat{\beta}'^G_1(\Delta,t) \geq \beta^d_{hc\_1}(\Delta,t),$$
$$\widehat{\beta}'^G_2(\Delta,t) \geq \beta^d_{hc\_2}(\Delta,t),$$

where

$$\widehat{\beta}'^G_1(\Delta,t) = bdf(\Delta,\tau^*(t)),$$
$$\widehat{\beta}'^G_2(\Delta,t) = RT(\widehat{\beta}'^G_1(\Delta,t),\alpha_{hc\_1}(\Delta,t)).$$

If a leaky bucket $lb_{hc\_1}(\Delta,t) = b_1(t) + r_1(t)\cdot\Delta$ is used to represent $\alpha_{hc\_1}(\Delta,t)$, then,

$$\begin{aligned}\widehat{\beta}'^G_2(\Delta,t) &= RT(bdf(\Delta,\tau^*(t)), lb_{hc\_1}(\Delta,t))\\ &= \sup_{0\leq\lambda\leq\Delta}\big\{\max\{0,\lambda-\tau^*(t)\} - b_1(t) - r_1(t)\cdot\lambda\big\}.\end{aligned}$$

As $\widehat{\beta}'^G_2(\Delta,t)\geq 0$, we abbreviate $\widehat{\beta}'^G_2(\Delta,t)$ as follows

$$\widehat{\beta}'^G_2(\Delta,t) = \max\big\{0,(1-r_1(t))\cdot\Delta-\tau^*(t)-b_1(t)\big\} \quad (10)$$

Then, to get the $\max(\tau^*(t))$, one only needs to keep the following two inequalities to be true

$$\max\{0,\Delta-\tau^*(t)\} \geq \beta^d_{hc\_1}(\Delta,t). \quad (11)$$
$$\max\big\{0,(1-r_1(t))\cdot\Delta-\tau^*(t)-b_1(t)\big\} \geq \beta^d_{hc\_2}(\Delta,t). \quad (12)$$

Both $\widehat{\beta}'^G_1(\Delta,t)$ and $\widehat{\beta}'^G_2(\Delta,t)$ are rate-latency functions. The backward derivation with the complex (de-)convolution is eliminated by computing the maximum $\tau^*(t)$ with the constraint of Eq. 11, 12. In this example, $\widehat{\beta}'^G_2(\Delta,t)$ of Eq. 10 is a closed-form equation w.r.t. $\tau^*(t)$ and the leaky rate and the bucket size of the leaky bucket, which will be proved in the following section.

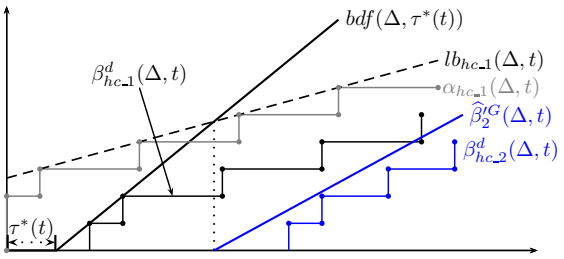


Figure 8: The scheme for showing how to compute $\tau^*(t)$ of two high-critical tasks

5.2.2 General case

Analogous to the computation of $\tau^*(t)$ with two high-critical tasks, for the system with more than two high-critical tasks, a similar procedure can be taken. As shown in Fig. 9, for the workload arrival curve of each task, a leaky bucket is used to conservatively represent it. Based on the leaky buckets and the assumption of LFII $\tau^*(t)$, by applying Eq. 3, one can get the resource guarantee service $\widehat{\beta}'^G_i$ for each task in a forward way. To meet the deadlines of each task, one only needs to keep the following inequalities to be true.

$$\widehat{\beta}'^G_i(\Delta,t) \geq \beta^d_{hc\_i}(\Delta,t),\ \forall i\in[1..n]. \quad (13)$$

However, to get $\widehat{\beta}'^G_i(\Delta,t)$, a step-by-step computation of applying Eq. 4 should be used, which blocks the computation speed of $\tau^*(t)$. To remove the step-by-step computation, a closed-form equation of $\widehat{\beta}'^G_i(\Delta,t)$ is derived by applying the following lemma.

LEMMA 1. *Suppose a mixed-criticality system modeled as Fig. 2. At time t, for each task, assume the* LFII *is $\tau^*(t)$, the deadline service demand is $\beta^d_{hc\_i}(\Delta,t)$, and the workload arrival curve is $\alpha_{hc\_i}(\Delta,t)$. If a leaky bucket with the form of $lb_{hc\_i}(\Delta,t) = r_i(t)\Delta + b_i(t)$ is used to conservatively represent $\alpha_{hc\_i}(\Delta,t)$, i.e., $lb_{hc\_i}(\Delta,t)\geq\alpha_{hc\_i}(\Delta,t)$, the resource guarantee service $\widehat{\beta}'^G_i(\Delta,t)$ for each task is as follows:*

$$\widehat{\beta}'^G_i(\Delta,t) = \max\{0,(1-R_i(t))\cdot\Delta-\tau^*(t)-B_i(t)\}, \quad (14)$$

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 proof this by induction.
If $n=1$,

$$\begin{aligned}\widehat{\beta}'^G_1(\Delta,t) &= \max\{0,(1-R_1(t))\cdot\Delta-\tau^*(t)-B_1(t)\}\\ &= \max\{0,\Delta-\tau^*(t)\}\\ &= bdf(\Delta,\tau^*(t)),\end{aligned}$$

which is true as shown in Fig. 9.

We assume that Eq. 14 is true for the task τ_{hc_n} ($n\neq 1$). Then, for the task $\tau_{hc\_(n+1)}$, by using the Real-Time Interface analysis,

$$\begin{aligned}\widehat{\beta}'^G_{n+1}(\Delta,t) &= RT(\widehat{\beta}'^G_n(\Delta,t), lb_{hc\_(n+1)}(\Delta,t))\\ &= \sup_{0\leq\lambda\leq\Delta}\big\{\max\{0,(1-R_n(t))\cdot\Delta-\tau^*(t)-B_n(t)\}\\ &\quad - b_{n+1}(t) - r_{n+1}(t)\cdot\lambda\big\}.\end{aligned}$$

As $\widehat{\beta}'^G_{n+1}(\Delta,t)\geq 0$, $\widehat{\beta}'^G_{n+1}(\Delta,t)$ can be rewritten as follows

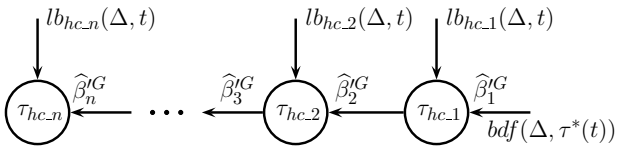


Figure 9: The flow of forward derivation for a FP system with multiple high-critical tasks

for brevity,

$$\begin{aligned}\widehat{\beta}_{n+1}'^G(\Delta,t) &= \max\big\{0,(1-R_n(t)-r_{n+1}(t))\cdot\Delta \\ &\quad -\tau^*(t)-(B_n(t)+b_{n+1}(t))\big\} \\ &= \max\big\{0,(1-R_{n+1}(t))\cdot\Delta-\tau^*(t)-B_{n+1}(t)\big\}\end{aligned}$$

□

Note that, to guarantee the system is schedulable, $1-R_i(t)$ in Eq. 14 should be greater than 0.

THEOREM 2. *By using the conservatively representation $lb_{hc\_i}(\Delta,t) \geq \alpha_{hc\_i}(\Delta,t)$ to compute $\widehat{\beta}_i'^G(\Delta,t)$, under the condition of Eq. 13, all high-critical tasks can meet their deadlines.*

PROOF. As $\widehat{\beta}_i'^G(\Delta,t) \geq \beta^d_{hc\_i}(\Delta,t)$ is true for all high-critical tasks, one only needs to prove that the actual service $\widehat{\beta}_i'^{ACT}(\Delta,t)$ is equal to or greater than $\widehat{\beta}_i'^G(\Delta,t)$. We proof this also by induction.

When $n=1$,

$$\widehat{\beta}_1'^{ACT}(\Delta,t) = bdf(\Delta,\tau^*(t)) = \widehat{\beta}_1'^G(\Delta,t).$$

We assume $\widehat{\beta}_n'^{ACT}(\Delta,t) \geq \widehat{\beta}_n'^G(\Delta,t)$ is true for the task τ_{hc_n} $(n \neq 1)$. Then, for the task $\tau_{hc\_(n+1)}$, we have

$$\begin{aligned}\widehat{\beta}_{n+1}'^{ACT}(\Delta,t) &= RT(\widehat{\beta}_n'^{ACT}(\Delta,t),\alpha_{hc\_n}(\Delta,t)) \\ &= \sup_{0\leq\lambda\leq\Delta}\big\{\widehat{\beta}_n'^{ACT}(\lambda,t)-\alpha_{hc\_n}(\lambda,t)\big\}.\end{aligned}$$

As $\widehat{\beta}_n'^{ACT}(\Delta,t) \geq \widehat{\beta}_n'^G(\Delta,t)$ and $lb_{hc\_n}(\lambda,t) \geq \alpha_{hc\_n}(\lambda,t)$, we have

$$\begin{aligned}\widehat{\beta}_{n+1}'^{ACT}(\Delta,t) &= \sup_{0\leq\lambda\leq\Delta}\big\{\widehat{\beta}_n'^{ACT}(\lambda,t)-\alpha_{hc\_n}(\lambda,t)\big\} \\ &\geq \sup_{0\leq\lambda\leq\Delta}\big\{\widehat{\beta}_n'^{G}(\Delta,t)-lb_{hc\_n}(\lambda,t)\big\} \\ &= \widehat{\beta}_{n+1}'^G(\Delta,t).\end{aligned}$$

Therefore, as $\widehat{\beta}_i'^{ACT}(\Delta,t) \geq \widehat{\beta}_i'^G(\Delta,t) \geq \beta^d_{hc\_i}(\Delta,t)$ is true for all tasks, all deadlines can be met. □

5.2.3 Leaky bucket representation

The closed-form equation $\widehat{\beta}_i'^G(\Delta,t)$ of Eq. 14 is a rate-latency function w.r.t. $R_i(t)$ and $B_i(t)$. With this closed-form rate-latency function, one only needs compute the maximum $\tau^*(t)$ with the constraint of n inequalities of Eq. 13. Now we show how to get parameters $R_i(t)$ and $B_i(t)$ during the runtime.

As described in Section 3.3, the future events are bounded by the prediction of using the dynamic counters. The future events bound, together with the backlogged events, provide the workload arrival curve and deadline service demand. Let $\mathcal{U}_{hc\_i}(\Delta,t)$ denote the future event arrival bound of high-critical task τ_{hc_i} at time t. To bound the future workload and service demand, the backlogged demand is also included.

DEFINITION 2. *(Backlogged Demand [8]) Suppose the set of unfinished events of S_{hc_i} in the buffer at time t are denoted as $\mathbf{E_i(t)}$. Let $D_{i,j}$ denote the absolute deadline for event $e_{i,j} \in \mathbf{E_i(t)}$ of the task τ_{hc_i}. A backlogged demand for stream S_{hc_i} is defined as*

$$B_{hc\_i}(\Delta,t) = c_i \cdot \begin{cases} (j-1), & D_{i,j}-t < \Delta \leq D_{i,j+1}-t, \\ |\mathbf{E_i(t)}|, & \Delta > D_{i,|\mathbf{E_i(t)}|}-t, \end{cases}$$

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

Hence, the workload arrival curve and deadline service demand for every high critical task at time t are:

$$\alpha_{hc\_i}(\Delta,t) = c_i \cdot \mathcal{U}_{hc\_i}(\Delta,t) + c_i \cdot |\mathbf{E_i(t)}|. \tag{15}$$

$$\beta^d_{hc\_i}(\Delta,t) = c_i \cdot \mathcal{U}_{hc\_i}(\Delta-D_i,t) + B_{hc\_i}(\Delta,t). \tag{16}$$

From the Eqs. 1, 2, 15, we know the composition of $\alpha_{hc\_i}(\Delta,t)$ 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 $\alpha_{hc\_i}(\Delta,t)$. But, in order to make our computation more tight, the leaky bucket should be as close to $\alpha_{hc\_i}(\Delta,t)$ as possible. Since the leaky bucket corresponding to the staircase function with the largest period in $\mathcal{U}_{hc\_i}$ is close to $\alpha_{hc\_i}(\Delta,t)$ in the long term, we use the staircase function with the largest period in $\mathcal{U}_{hc\_i}$ to compose a leaky bucket to represent the workload arrival curve $\alpha_{hc\_i}(\Delta,t)$. As the period of each staircase function in a task 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 WCET/(largest period). Hence, $1-R_i(t)$ in Eq. 14 is also known offline and fixed during the runtime. Then, to get $\widehat{\beta}_i'^G(\Delta,t)$, one only needs to know the bucket size $b_i(t)$. Suppose for the task τ_{hc_i}, $DC_i^{\#}$ is the counter for tracking the staircase function with the largest period δ_i^{max} in Algo. 1. At time t, it can be derived from Eqs. 1, 2, 15 that

$$b_i(t) = c_i\cdot|\mathbf{E_i(t)}|+c_i\cdot\begin{cases} DC_i^\sharp + \frac{t-k_i^\sharp\cdot\delta_i^{max}}{\delta_i^{max}} & \text{if } DC_i^\sharp < N_i^{\sharp u} \\ DC_i^\sharp & \text{if } DC_i^\sharp = N_i^{\sharp u} \end{cases} \tag{17}$$

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. 17. $\widehat{\beta}_i'^G(\Delta,t)$ can also be conveniently obtained as $R_i(t)$ is fixed and $B_i(t)$ is easy to obtain with the support of Eq. 17.

5.2.4 Deriving LFII

To solve the maximization problem with the constraint of Eq. 13, the first step is to use the current parameters in monitors to update the $\widehat{\beta}_i'^G(\Delta,t)$ and $\beta^d_{hc\_i}(\Delta,t)$, as shown in Fig. 7. To solve the inequality of $\widehat{\beta}_i'^G(\Delta,t)$ and $\beta^d_{hc\_i}(\Delta,t)$, one needs to compare $\widehat{\beta}_i'^G(\Delta,t)$ with $\beta^d_{hc\_i}(\Delta,t)$. As $\Delta = +\infty$, it is impossible to compare $\widehat{\beta}_i'^G(\Delta,t)$ with $\beta^d_{hc\_i}(\Delta,t)$ in whole $\Delta^{+\infty}$. Actually, in this comparison, only a limited segment of Δ needs to be compared. If $\widehat{\beta}_i'^G(\Delta,t) \geq \beta^d_{hc\_i}(\Delta,t)$ in this limited segment, $\widehat{\beta}_i'^G(\Delta,t) \geq \beta^d_{hc\_i}(\Delta,t)$ in any interval Δ.

In a schedulable system, suppose for a high-critical task τ_{hc_i}, at time t, there is $|\mathbf{E_i(t)}|$ 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. 10. c_i is the WCET for this high-critical task τ_{hc_i}.

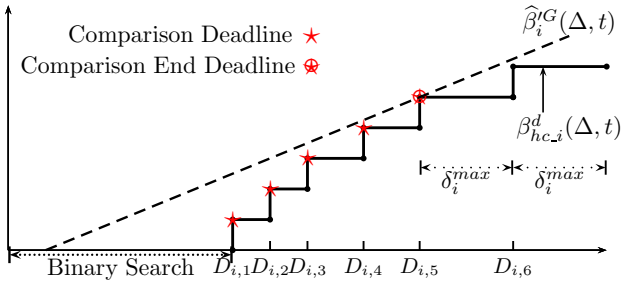


Figure 10: An illustration how to do the comparison

LEMMA 2. *If $D_{i,x+1} - D_{i,x} = \delta_i^{max}$ and $x > |\mathbf{E_i(t)}|$, then for the absolute deadline of k-th event ($k \geq x$), we have*

$$D_{i,k+1} - D_{i,k} = \delta_i^{max}. \tag{18}$$

PROOF. From Eq. 16, we know the deadline trace is decided by $\mathcal{U}_{hc\_i}(\Delta - D_i, t)$ and $B_{hc\_i}(\Delta, t)$. As $x > |\mathbf{E_i(t)}|$, the absolute deadline for x-th event only depends on $\mathcal{U}_{hc\_i}(\Delta - D_i, t)$. $\mathcal{U}_{hc\_i}(\Delta - D_i, t)$ 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 $\mathcal{U}_{hc\_i}(\Delta - D_i, t)$ only depends on the staircase function with the largest period. As $\mathcal{U}_{hc\_i}(\Delta - D_i, t)$ is convex, for the k-th event ($k \geq x$), $\mathcal{U}_{hc\_i}(\Delta - D_i, t)$ also depends on the staircase function with the largest period. Hence, $D_{i,k+1} - D_{i,k} = \delta_i^{max}$. □

LEMMA 3. *Suppose the task τ_{hc_i} is schedulable with a rate-latency function $\beta(\Delta)$. For the x-th and k-th event in Lem. 2, if $\beta(D_{i,x}) \geq \beta^d_{hc\_i}(D_{i,x}, t) \geq 0$, we have $\beta(D_{i,k}) \geq \beta^d_{hc\_i}(D_{i,k}, t)$.*

PROOF. Assume $\beta(\Delta) = \max\{0, r \cdot \Delta + b\}$, as $\beta(D_{i,x}) \geq \beta^d_{hc\_i}(D_{i,x}, t) \geq 0$, i.e.,

$$r \cdot D_{i,x} + b \geq \beta^d_{hc\_i}(D_{i,x}, t),$$
$$r \cdot D_{i,x} + b + (k-x) \cdot c_i \geq \beta^d_{hc\_i}(D_{i,x}, t) + (k-x) \cdot c_i.$$

From Lem. 2, as $D_{i,k+1} - D_{i,k} = \delta_i^{max}$, $\beta^d_{hc\_i}(D_{i,k+1}, t) - \beta^d_{hc\_i}(D_{i,k}, t) = c_i$. We have

$$\beta^d_{hc\_i}(D_{i,k}, t) = \beta^d_{hc\_i}(D_{i,x}, t) + (k-x) \cdot c_i.$$

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

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

Therefore, $\beta(D_{i,k}) = r \cdot D_{i,k} + b \geq \beta^d_{hc\_i}(D_{i,k}, t)$. □

Lem. 3 indicates that, if a rate-latency function $\beta(\Delta) \geq \beta^d_{hc\_i}(\Delta, t)$ within a interval of $[0, D_{i,x}]$, $\beta(\Delta)$ will be greater than $\beta^d_{hc\_i}(\Delta, t)$ in any interval. In this paper, we define the earliest deadline that satisfies Eq. 18 as the comparison end deadline, as shown in Fig. 10. With the Lem. 3, to do the comparison of Eq. 13, one only needs to compare $\widehat{\beta}_i'^G(\Delta, t)$ with $\beta^d_{hc\_i}(\Delta, t)$ before the comparison end deadline. For example, as shown in Fig. 10, there are only 5 deadlines before the comparison end deadline. If $\widehat{\beta}'^G_{hc\_i}(D_{i,j}, t)$ is greater than $\beta^d_{hc\_i}(D_{i,j}, t)$ in these 5 deadlines, this rate-latency function in other deadlines is also greater than the $\beta^d_{hc\_i}(D_{i,j}, t)$. As $0 \leq \tau^*(t) \leq D_{i,1}$, we use the binary search to get the maximum $\tau^*(t)$. The computing complexity for one high-critical task is $O(log(m))$.

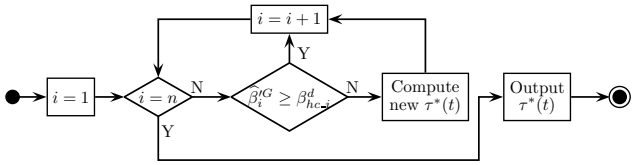


Figure 11: The program diagram to compute the maximum $\tau^*(t)$ with the constraint of n inequalities

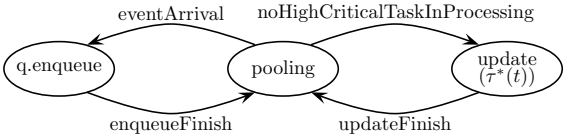


Figure 12: Finite state transitions of our shaper

In short, for our proposed lightweight scheme, only n inequalities need to be solved. As we only need to get the maximum $\tau^*(t)$ that makes all inequalities hold, it is not necessary to compute $\tau^*(t)$ for every inequality. The bubble sorting with one iteration can be used to pick out the maximum $\tau^*(t)$. It works like this, as shown in Fig. 11, we start the searching with a very large $\tau^*(t)$. As this large $\tau^*(t)$ will make $\widehat{\beta}_1'^G < \beta^d_{hc\_1}$, a new $\tau^*(t)$ will be computed by solving $\widehat{\beta}_1'^G \geq \beta^d_{hc\_1}$. This $\tau^*(t)$ is used in the second inequality. If the rate-latency function $\widehat{\beta}_2'^G$ with $\tau^*(t)$ is greater than $\widehat{\beta}^d_{hc\_2}$, this $\tau^*(t)$ is used in the third inequality. If not, we compute a new $\tau^*(t)$ of the second inequality, and use this new $\tau^*(t)$ in the third inequality. $\tau^*(t)$ is computed in this way till the last inequality. The computed $\tau^*(t)$ is the maximum LFII that satisfies all inequalities. The whole computing complexity is $O(n \cdot log(m))$.

5.3 Shaping Runtime Behavior

Algorithm 2 Shape the incoming flow of low-critical tasks

Input: signal s;
1: **if** s = event_arrival **then**
2: $q_i.enqueue()$ ▷ store event
3: **end if**
4: **if** s = noHighCriticalTaskInProcessing **then**
5: update(τ^*) ▷ update the τ^*
6: **end if**
7: **while** $\min(q_i.length) > 0 \wedge \tau^* > 0$ **do**
8: **for** $i = 1 \rightarrow n$ **do** ▷ pooling to find the right event
9: **if** $c_i \leq \tau^*$ **then**
10: $q_i.dequeue()$ ▷ send event
11: $\tau^* = \tau^* - c_i$ ▷ reduce τ^* by WCET
12: break
13: **end if**
14: **end for**
15: **end while**

The shaper controls the workload inflow of low-critical tasks based on the LFII $\tau^*(t)$. In general, we have to decide when to update the LFII $\tau^*(t)$ and how to regulate the workload inflow of low-critical tasks. Our shaper is designed as a finite state machine, as shown in Fig. 12. Normally, our shaper is in the pooling state. There are two signals that interrupt the pooling state. One signal is noHighCriticalTaskInProcessing, which means there is no high-critical tasks that are being processed in the processor.

Table 1: Event streams according to [4]

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
p[ms]	198	102	283	354	239	194	148	114	313	119
j[ms]	387	70	269	387	222	260	91	13	302	187
d[ms]	48	45	58	17	65	32	78	-	86	89
w[ms]	12	7	7	11	8	5	13	14	5	6

Table 2: The set of high-critical event streams

Sets	Streams	Utilization	Sets	Streams	Utilization
Set 1	S_3,S_8,S_2	0.2162	Set 3	S_8,S_3,S_7,S_2,S_1,S_6	0.3904
Set 2	S_5,S_3,S_8,S_9,S_4,S_2	0.2967	Set 4	S_{10},S_7,S_5,S_8,S_9, S_2,S_4,S_3,S_1	0.4956

This signal is used to interrupt the pooling state to the state of update($\tau^*(t)$). The reason for using this signal as the updating signal is that, the deadline service demand of Eq. 16 assumes that there are no pending high-critical tasks that are being processed. Therefore, to update LFII $\tau^*(t)$, in the processor, there also should be no high-critical tasks that are being processed. Otherwise, the computed LFII $\tau^*(t)$ will not be accurate.

Another signal is the low-critical event arrival. When the low-critical event arrives, it is first stored in a queue in the shaper. The workflow of our shaper is shown in Algo. 2. For every event stream, a queue is used in the shaper to store them. The shaping policy is that, any event is first stored in a queue. The shaper pools the queue to find the candidate events whose WCETs are less than $\tau^*(t)$, and picks up the highest-priority event from the candidates to send it to the processor, as shown in line 2, 10 in Algo. 2.

6. IMPLEMENTATION AND EVALUATION

In this section, we evaluate the proposed runtime shaping method and compare its shaping performance with the offline shaping. The simulator is implemented in MATLAB by applying MPA and RTC/S tools from [24]. All the results are obtained from a simulation host with Intel Q9300 processor and 5 GB RAM.

6.1 Evaluation Setup

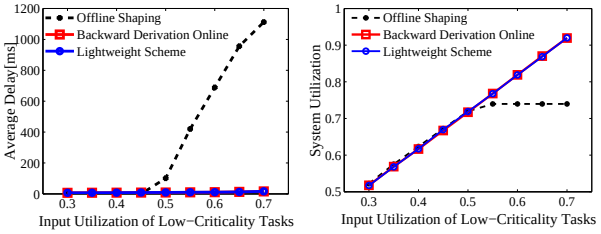
We use the system shown in Fig. 2 for our experiments. The model contains a set of low-critical tasks and a set of high-critical tasks. The event streams for high-critical tasks are taken from [4], which are shown in Tab. 1. For the evaluations, 4 sets of high-critical event streams are chosen, as shown in Tab. 2. The utilizations of the 4 sets are also shown in Tab. 2. The relative deadline D_i of a stream S_i is set equal to its period. The low-critical tasks are pseudo-random testcases, which are five event streams generated by using UUniFast [2]. The low-critical tasks are independent from each other, and are activated randomly, while their mean interarrival times were chosen, such that together they impose an additional utilization varying from 0.3 to 0.7. We don't set the deadlines for low-critical events, and don't drop any events. Note that, if the WCET of low-critical tasks is larger than the largest LFII, their events will be blocked forever.

In this work, three shaping performances are evaluated, as shown in the following

- Shaping by offline computed bound.
- Shaping by backward derivation online.
- Shaping by our proposed lightweight scheme.

For every simulation, we simulate 10 seconds for 100 times, and count the delay of low-critical events from the time of entering into the queue to the time of going out from the queue. We use the average delay as a metric of showing the shaping performance. The system utilization is also recorded, which indicates how long the processor is busy with processing tasks in 10 seconds.

6.2 Evaluation Results



(a) Average Delay (b) System utilization

Figure 13: Performance evaluation of online shaping and offline shaping

Firstly, we show the performance difference of the online shaping and the offline shaping. For this comparison, set 1 is chosen as inputs of high-critical tasks. The utilization of low-critical tasks varies from 0.3 to 0.7. Fig. 13(a) shows the average delay by using the online shaping with the backward derivation and the lightweight scheme, and the offline shaping. From it, we can see that, when the input utilization of low-critical tasks is smaller than 0.45, the delay keeps a small value for all shapings. But when the input utilization is greater than 0.45, the average delay of offline shaping increases sharply. This is because this utilization exceeds the largest utilization that this shaping bound can reach. For the online shaping, the delay keeps small and increases slowly. Fig. 13(b) shows the system utilization. It can be seen that the highest system utilization of the offline shaping can only reach about 0.75, and the highest system utilization of online shaping can reach 0.9. Besides, the two subfigures show that the shaping effect with the lightweight scheme is the same for the online shaping with the backward derivation.

Secondly, we observed the influence of high-critical tasks on the online shaping performance. Four sets of event streams with the utilization of about 0.2, 0.3, 0.4, 0.5 are used as the inputs for activating high-critical tasks. By varying the imposed utilization of low-critical tasks, the event delay is compared based on the same system utilization for the four sets. The results are shown in Fig. 14. The left axis shows the delay per event, and the right axis shows the difference ratio (lightweight scheme/backward derivation) between the lightweight scheme and the backward-derivation method. From the four figures, we can see the average event delay increases with an increasing of system utilization. It can also be seen that there is little difference in the online shaping between with the backward-derivation method and with the lightweight scheme if the system utilization is below 0.8.

Lastly, the computation time of updating the LFII $\tau^*(t)$ is compared with the backward-derivation method and the lightweight scheme. For the backward-derivation method, the program will overflow if workload prediction is used without any approximation. To make the backward derivation work, the interception of workload prediction

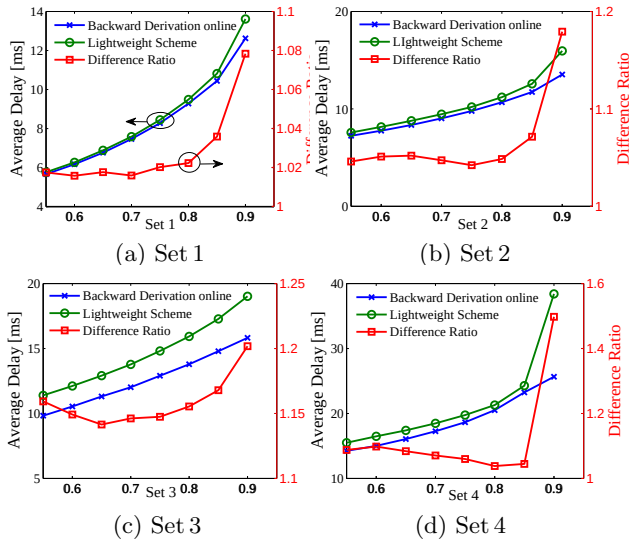


Figure 14: The event delay with system utilization

with a length of 10 relative deadlines is used. Fig. 15 presents the worst, best, and average case computational expenses of updating τ^* with respect to the number of high-critical event streams. For the backward-derivation method, the average computation time for one stream is 7.9 ms, and increases linearly with the number of streams. For the lightweight scheme, the average computation time for one stream is only 36.1 μs, which is two orders of magnitude lower than the computation time by using the backward-derivation method. This figure also shows that, even for 10 streams, the average computation time of using the lightweight scheme is only 164.7 μs. The result indicates that the timing overhead of the lightweight scheme is considerably smaller.

7. CONCLUSIONS

This paper presents an approach to adaptively shape the inflow workload of low-critical tasks based on the actual demand of the high-critical tasks at runtime. Compared to the shaping with the offline bounds, the low-critical event delay is reduced, and the system utilization is improved. Since our adaptation method is a lightweight scheme with the complexity of $O(n \cdot log(m))$, our approach is suitable for online application to update the shaping bound. The experiments also demonstrate the efficiency and effectiveness of shaping scheme, and it also shows that the timing overhead with our proposed lightweight scheme is two orders of magnitude lower than using the backward-derivation method.

8. ACKNOWLEDGMENTS

This work has been partly funded by China Scholarship Council, German BMBF projects ECU (grant number: 13N11936) and Car2X (grant number: 13N11933).

9. REFERENCES

[1] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *RTSS*, pages 34–43. IEEE, 2011.

[2] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[3] A. Burns and R. Davis. Mixed criticality systems: A review. *University of York, Tech. Rep*, 2013.

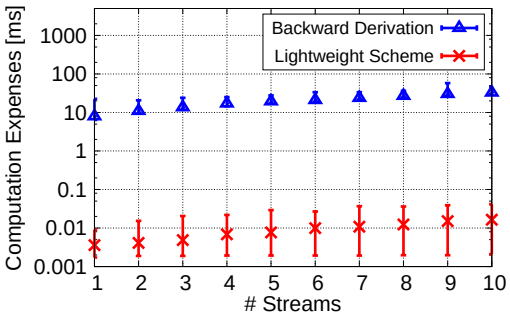


Figure 15: Computation time of updating the Lfii

[4] A. Hamann and R. Ernst. Tdma time slot and turn optimization with evolutionary search techniques. In *DATE*, pages 312–317. IEEE, 2005.

[5] B. Hu, K. Huang, G. Chen, and A. Knoll. Evaluation of runtime monitoring methods for real-time event streams. In *ASPDAC*, pages 582–587. IEEE, 2015.

[6] K. Huang, G. Chen, C. Buckl, and A. Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *DAC*, pages 430–436, 2012.

[7] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *RTSS*, pages 23–32. IEEE, 2009.

[8] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.

[9] B. James, B. Todd, B. Pam, H. Jon, P. James, S. Prakash, S. John, S. Peter, S. Douglas, and U. Russell. A research agenda for mixed-criticality systems.

[10] K. Lampka, K. Huang, and J.-J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. CODES+ISSS, pages 267–276, 2011.

[11] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems. EMSOFT, pages 107–116, 2009.

[12] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[13] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *RTSS*, pages 88–96, Dec 2013.

[14] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring arbitrary activation patterns in real-time systems. In *RTSS*, pages 293–302, Dec 2012.

[15] M. Neukirchner, S. Quinton, R. Ernst, and K. Lampka. Multi-mode monitoring for mixed-criticality real-time systems. CODES+ISSS, pages 1–10, Sept 2013.

[16] M. Neukirchner, S. Quinton, T. Michaels, P. Axer, and R. Ernst. Sensitivity analysis for arbitrary activation patterns in real-time systems. In *DATE*, pages 135–140, March 2013.

[17] L. Phan and I. Lee. Improving schedulability of fixed-priority real-time systems using shapers. In *RTETAS*, pages 217–226, April 2013.

[18] J. A. Rowson and A. Sangiovanni-Vincentelli. Interface-based design. In *DAC*, pages 178–183. ACM, 1997.

[19] S. Tobuschat, M. Neukirchner, L. Ecco, and R. Ernst. Workload-aware shaping of shared resource accesses in mixed-criticality systems. In *CODES+ISSS*, page 35. ACM, 2014.

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

[21] E. Wandeler. *Modular performance analysis and interface-based design for embedded real-time systems*. PhD thesis, ETH Zurich, Sept 2006.

[22] E. Wandeler, A. Maxiaguine, and L. Thiele. On the use of greedy shapers in real-time embedded systems. *ACM Trans. Embed. Comput. Syst.*, 11(1):1:1–1:22, Apr. 2012.

[23] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *EMSOFT*, pages 80–89, 2005.

[24] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.