Improving Schedulability of Fixed-Priority Real-Time Systems using Shapers[*†]

Linh T.X. Phan Insup Lee
Department of Computer and Information Sciences, University of Pennsylvania
Email: {linhphan, lee}@cis.upenn.edu

***Abstract*—In this paper, we introduce a technique for improving the schedulability of real-time embedded systems with fixed-priority scheduling. Our technique uses *shapers* to reduce the resource interference between higher-priority and lower-priority tasks, and thus enables more lower-priority tasks to be scheduled. We present a closed-form solution for the optimal greedy shaper for periodic tasks with jitter, as well as a schedulability condition for tasks in the presence of shapers. We also discuss two applications of greedy shapers: In compositional scheduling frameworks, shapers can help optimize the resource interfaces of real-time components, and in mixed-criticality systems, they can reduce deadline misses of low-criticality tasks while preserving schedulability of high-criticality tasks, even with lower priorities. We demonstrate the utility of our technique through an evaluation based on randomly generated workloads.**

I. Introduction

For the past several decades, fixed-priority (FP) scheduling has been one of the most commonly used scheduling algorithms in safety-critical real-time embedded systems, partly due to its flexibility, its simple run-time mechanism, and its small overhead [11]. However, in a fixed-priority system, low-priority tasks often experience resource interference from higher-priority tasks, which leads to long response times (compared to systems that use dynamic schedulers, such as Earliest Deadline First) and requires more resources to guarantee that the system is schedulable. Hence, reducing interference from high-priority tasks is critical to improving the schedulability and optimizing the resource requirements of the system. In fully preemptive FP, minimizing the interference due to preemption also helps reduce cache-related overheads and energy consumption [7].

Several techniques for improving resource interference in FP systems have been proposed (see e.g., [5]–[7], [13], [23], [24]); however, most of these techniques assume a strictly periodic task model. This assumption is overly conservative for many reactive embedded systems, where tasks often exhibit *jitter*, i.e., deviations from true periodicity. This jitter typically comes from the release-delay overheads induced by tick-driven scheduling [18], execution of interrupt service routines [4], or I/O overheads. Other sources of jitter include delays caused by scheduling, data dependencies, and communication, especially in a distributed setting. For example, in ARINC avionics systems [2], tasks in different scheduling partitions are connected over a switched Ethernet; due to network delay, tasks in a partition are not always released strictly periodically, but with a certain jitter. Automotive networks are another example: here, sensor data are sampled at a constant rate, but the results often pass through a series of electronic control units (ECUs) and the bus before arriving at a given ECU. Since the resource of each component is often shared among multiple tasks, scheduling and task execution within a component can delay the input data for a variable amount of time, which results in jittery output. If this output is then used to activate the successor task in the next component, this task will itself be released in a jittery manner or (when the jitter exceeds the length of a period) even in bursts.

One simple approach to dealing with jitter is to transform each task with a jittery release pattern into a new periodic/sporadic task with a shorter period [4]. Existing interference reduction techniques for periodic tasks (e.g., [5]–[7], [13], [23], [24]) can then be applied to the transformed task sets. While this method is safe, the transformation can lead to overly pessimistic schedulability analysis results. In addition, the resulting smaller periods (of higher-priority tasks) exacerbate the resource interference with lower-priority tasks, which can diminish or even cancel out the benefits of the interference reduction techniques. Further, these techniques modify the behavior of FP to reduce the preemption overheads, which often requires fine-grained information about the tasks' code structure or complex computation of preemption attributes (e.g., [6], [23], [24]). A technique that does not change the FP behavior or tasks' attributes was proposed in [17]; however, it requires the use of CPU frequency scaling, which may not be available in all systems.

In this paper, we propose a technique based on shaping [3] to improve the schedulability of FP systems that contain periodic tasks with jitter. Our key observation is that, in settings where tasks exhibit jitter, the resource interference of higher-priority tasks on the lower-priority ones is much higher than in a strictly periodic setting. Typically, the larger the jitter of a higher-priority task, the higher the resource interference it imposes on lower-priority tasks. By using shapers to reduce the maximum jitter/burst, we can reduce the interference, and thus enable more lower-priority tasks to be scheduled. In this paper, we focus on the use of *greedy shapers*, since they have a simple representation (as arrival curves [3]), and they preserve the worst-case output arrival patterns and can therefore be combined with component-based analysis frameworks for distributed architectures (e.g., [22]). Our technique based on greedy shaping also preserves both the fixed-priority scheduler and the tasks' parameters, so it can be

[*]This research was supported in part by the ARO grant W911NF-11-1-0403, NSF grants CNS-1117185 and CPS-1135630, and MKE (The Ministry of Knowledge Economy), Korea, under the Global Collaborative R&D program supervised by the KIAT (M002300089).

[†]This is the author's version, which fixes an error in Example 2 in the official IEEE version.

integrated into many existing fixed-priority systems (e.g., with Deadline Monotonic and $(D-J)$-monotonic [25] schedulers).

Greedy shapers were first introduced in the context of communication networks [3]. They are commonly used to ensure that packets are sent at regular intervals rather than in bursts, which results in lower queueing delays. In this setting, schedulability or interference with lower-priority flows are not primary concerns, so the choice of shaping functions is straight-forward: a greedy shaper can use any function that is at least equal to the arrival function of the source flow [3]. In safety-critical real-time embedded systems, however, schedulability is crucial, so better shaping functions are needed to meet the system's timing and resource constraints.

Contributions. This paper makes the following contributions:

- We propose the use of shaping to reduce resource interference between tasks with jittery release patterns, which improves schedulability and minimizes the total resource demands of the system (Section III).
- We introduce a method for computing the optimal shaper for the periodic-with-jitter task model (which minimizes a task's resource interference on lower-priority tasks without affecting its schedulability), and we integrate shaping into the schedulability condition (Sections IV and V).
- We discuss two applications of shapers. First, we show how shapers can be integrated with compositional scheduling frameworks, and we analyze their effect on the components' interfaces (Section VI). Second, we show that in mixed-criticality systems, shapers can preserve schedulability of high-criticality tasks while minimizing deadline misses of low-criticality tasks (Section VII).
- We validate our technique with an evaluation using a variety of real-time workloads. The results show that our proposed technique can reduce the system's deadline miss ratio by up to 35%, and it reduces components' interface bandwidth by up to 33% compared to a reference system without shapers (Section VIII).

Related work. To the best of our knowledge, this work is the first to use shaping to reduce resource interference in FP systems. Shaping was originally proposed for traffic shaping in communication networks [3]. Wandeler et al. [19], [20] extended it to the real-time domain but analyzes the end-to-end delay and buffer requirements of existing shapers, rather than designing new ones. Other work has designed shapers for optimizing specific metrics; for instance, Kumar et al. [9] showed how to compute a shaper that optimizes on-chip peak temperature. However, Kumar's approach to shaper design is specific to the EDF scheduling, and is not applicable in an FP setting for two reasons. First, optimality is proven assuming that the processor is fully available to the output jobs of the shaper; this is not the case in FP systems, where lower-priority tasks do not receive the full capacity of the processor. Second, Kumar uses the shaper to shape the combined demand of all tasks in the system, but there is currently no technique for computing the total demand of tasks with different deadlines and WCETs that are scheduled under FP.

There is other work that has some similarities to shaping. Sun and Liu [16] introduced the release guard protocol to release subtasks of an end-to-end task periodically, which is a special form of greedy shaper. Richter et al. [14] proposed an event adaptation function (EAF), which transforms an event stream into another that conforms to a given model. If an EAF is given a periodic model, it effectively acts as a shaper; the key challenge we solve in this paper is to design the model such that it improves schedulability.

At a high level, shapers are also similar to servers, which are commonly used to schedule aperiodic tasks together with periodic tasks [12]. Servers do not guarantee any properties by themselves, other than a limit on how much of the resource is used by the tasks they encapsulate; in particular, they do not consider the schedulability of the tasks they contain. Servers are a general concept, and in theory one could design a server that provides the same properties as a shaper, but then the server would effectively *become* a shaper, and its design would require an analysis similar to the one we propose here.

II. Background

Task model. The system consists of N independent periodic tasks with jitter (see e.g., [1], [18]), which are scheduled under preemptive FP on a uniprocessor. Each task T_i is characterized by (P_i, J_i, E_i, D_i), with $0 < E_i \leq \min(P_i, D_i)$, where P_i, J_i, E_i and D_i are non-negative real numbers that represent the task's period, jitter, worst-case execution time (WCET) and relative deadline. The jitter J_i is the maximum delay between the virtual ideal periodic release time of a job (of T_i) and its actual release time. All tasks have hard deadlines; the absolute deadline of a job of T_i is its actual release time plus D_i.

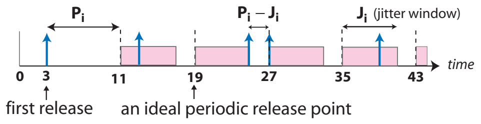


Fig. 1: A sample release pattern of T_i with $P_i = 8$, $J_i = 6$.

Figure 1 illustrates a sample release pattern of a task. We assume that the scheduler maintains a ready queue for each task, which contains all jobs of the task that are ready for scheduling, and we write $HP(T_i)$ (resp. $LP(T_i)$) to denote the set of all tasks with a higher (resp. lower) priority than T_i.

Modeling resource demands. The release patterns of a task can be modeled as an *arrival function*, which specifies the maximum number of jobs that are released over any (left-open or right-open) time interval of length t, for all $t \geq 0$. Let $\mathbb{R}_+$ be the set of non-negative real numbers. The arrival function of a task T_i with period P_i and jitter J_i is $\alpha_i : \mathbb{R}_+ \to \mathbb{N}$, where

$$\alpha_i(0) = 0, \text{ and } \alpha_i(t) = \left\lceil \frac{t + J_i}{P_i} \right\rceil, \forall\, t > 0. \qquad (1)$$

The number of execution units requested by a task over any time interval of a given length can then be modeled by its *request bound function* (RBF). The RBF of a task T_i with arrival function $\alpha_i(t)$ and WCET E_i is given by:

$$rbf_i(t) = E_i \cdot \alpha_i(t), \forall\, t \geq 0. \qquad (2)$$

Similarly, the minimum execution units that must be given to a task to ensure its schedulability is modeled by a *demand bound function* (DBF), which specifies, for each $t \geq 0$, the maximum number of execution units required by all jobs that are released and that have deadlines within an interval of length t. The DBF of a task T_i with relative deadline D_i and RBF $rbf_i(t)$ is:

$$dbf_i(t) = \max\{0,\ rbf_i(t - D_i)\},\ \forall\, t \geq 0.$$

We note that arrival functions and RBFs of periodic-with-jitter tasks are *good* functions, i.e., they are sub-additive and equal to zero at $t = 0$. Recall that f is sub-additive iff $f(t) \leq f(t-s) + f(s)$ for all $0 \leq s < t$.

Modeling resource availability. The resource supplied by a processor can be modeled by a *supply bound function* (SBF), denoted by $sbf(t)$, which gives the minimum number of execution units available over any time interval of length t, for all $t \geq 0$. SBFs are also known as *service functions* (possibly, after scaling to a task's WCET) [3].

Consider a task T that is processed by a processor with SBF f. Suppose g is the RBF of T, then the SBF of the minimum remaining resource after processing T is given by [3]:

$$\mathsf{Remain}(f,g)(t) \overset{\text{def}}{=} \sup\{f(x) - g(x) \mid 0 \leq x \leq t\}. \quad (3)$$

Further, if f' is the SBF of the remaining resource, then the maximum value for the RBF of T is given by [21]:

$$\mathsf{RTInverse}(f',f)(t) \overset{\text{def}}{=} f(t+\delta_t) - f'(t+\delta_t) \quad (4)$$

where $\delta_t = \sup\{\varepsilon \mid f'(t+\varepsilon) = f'(t)\}$.

Finally, let $\mathcal{F}$ be the set of increasing functions $f : \mathbb{R}_+ \to \mathbb{R}_+$. We define the $(\min,+)$ convolution, deconvolution and sub-additive closure of any $f, g \in \mathcal{F}$ as follows. For all $t \geq 0$:

$$(f \otimes g)(t) \overset{\text{def}}{=} \inf\{f(s) + g(t-s) \mid 0 \leq s \leq t\}; \quad (5)$$

$$(f \oslash g)(t) \overset{\text{def}}{=} \sup\{f(t+u) - g(u) \mid u \geq 0\}; \quad (6)$$

$$\mathsf{closure}(f) = \min\{f,\ f \otimes f,\ f \otimes f \otimes f, \ldots\}. \quad (7)$$

It follows that $\mathsf{closure}(f)$ is the maximum good function upper bounded by f; $(f \oslash f)$ is a good function; and $f \oslash f \geq f$. Finally, we say f is larger (resp. smaller) than or equal to g iff $f(t) \geq g(t)$ (resp. $f(t) \leq g(t)$) for all $t \in \mathbb{R}_+$.

III. Overview of our shaping approach

Basic idea: Recall our assumption (from Section II) that the scheduler maintains a separate ready queue for each task. The idea is to optimize the resource usage of the system by controlling these queues in such a way that (i) the interference between jobs from different queues is minimized, and (ii) the total resource requirements of all jobs within the system is reduced without violating the jobs' schedulability.

To achieve this, we insert a greedy shaper in front of each ready queue to shape the arrival pattern of jobs that are ready to be scheduled. The shaper can delay an already released job for a certain amount of time so as to bound the resource demands (requests) of the corresponding task to a desired level. Effectively, each shaper reduces its task's resource interference with the lower-priority tasks. At the same time, by postponing the execution of a higher priority job just long enough so that it can still meet its deadline, a shaper indirectly enables the scheduler to give resources to the lower-priority jobs, which improves their schedulability.

System architecture: Figure 2 illustrates the architecture of a system that implements the shaping technique. Each shaper σ_i has an input buffer B_i, which is used to hold newly released jobs before putting them into the ready queue Q_i. The scheduler does not need to be aware of the shapers – it can schedule the jobs in the ready queues exactly as if there were no shapers. We say that a shaper is *feasible* if it preserves schedulability of all tasks in the system.

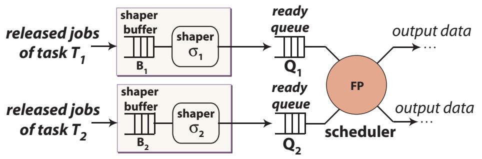


Fig. 2: System architecture for a system with shapers.

Greedy shapers. Informally, a greedy shaper [3] forces its output jobs to conform to a specific arrival function, the *shaping function*, and it outputs the jobs as soon as the shaping function allows it. In Figure 2, σ_i is the arrival function used by T_i's greedy shaper. For convenience, we also use σ_i to indicate the greedy shaper itself, and we use the terms "shaping function" and "shaper" interchangeably. Thus, the shaper σ_i delays each released job in the buffer B_i whenever placing the job to Q_i would violate the shaping function σ_i.

Formally, suppose the shaper σ_i receives job k of T_i at time t_k (i.e., t_k is the release time of job k), where $0 \leq t_k \leq t_{k+1}$ for all $k \geq 1$. Then, it will output job k to Q_i at time $t'_k = t_k + d_k$, where $d_k = 0$ if $k = 1$, and otherwise

$$d_k = \min\{\Delta \mid \forall 1 \leq j < k :\ k - j + 1 \leq \sigma_i(t_k + \Delta - t'_j + \varepsilon)\}$$

for an infinitely small positive value of ε. Intuitively, d_k is the minimum time that the shaper must delay job k to prevent its ready time from violating σ_k; this is obtained by ensuring that the number of ready jobs in every interval $[t'_j, t'_k]$ is no more than $\sigma_i(t'_k - t'_j + \varepsilon)$ for all $j < k$.

Example 1: Consider a task T_1 with period $P_1 = 4$ and jitter $J_1 = 3$, and suppose σ_1 is a periodic arrival function with the same period as the task, i.e., $\sigma_1(t) = \lceil t/4 \rceil$, for all $t \geq 0$. Then the shaper will output at most $\lceil t/4 \rceil$ jobs to the ready queue over any interval of length t, or at most one ready job every four time units. Figure 3 shows a release pattern of T_1 and the corresponding ready pattern after shaping.

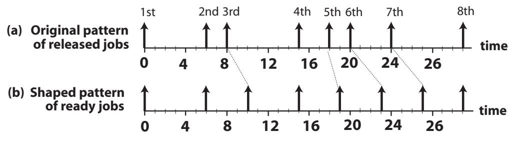


Fig. 3: Effect of the shaper $\sigma_1(t) = \lceil t/4 \rceil$ on $T_1 = (4,3,1,4)$.

In this pattern, due to jitter, jobs 3, 5 and 6 are released less than 4 time units after their predecessors. Since the shaper cannot allow more than one job to become ready within a 4-unit interval, it delays these jobs; in the figure, this is indicated by the dashed lines connecting release instants and the corresponding (delayed) ready instants. Note also that the ready pattern is less bursty and closer to a periodic pattern than the original release pattern, and that the delay imposed by the shaper is bounded.

In this paper, we assume that the shaper buffer is empty at time 0, and it is is large enough to prevent job loss. In our setting, this always holds because tasks are not released before time 0, and the number of active jobs per task at any instant is finite (otherwise, the task cannot be schedulable). In the rest of the paper, the term "shaper" refers to "greedy shaper".

IV. Optimal shaper computation

Our next goal is to compute an optimal greedy shaper for each task T_i in the system, i.e., one that minimizes the maximum response times of the lower-priority tasks. We begin by introducing some basic properties of greedy shapers.

Theorem 4.1: Let E_i, α_i and σ_i be the WCET, arrival function and shaper of T_i, where σ_i is a good function. Then,

1) the jobs output by the shapers (i.e., the ready jobs of Q_i) are bounded by the arrival function $\alpha_i^{gs} = \alpha_i \otimes \sigma_i$;
2) the greedy shaper serves as a virtual resource that offers a supply-bound function equal to $E_i\sigma_i$; and
3) if $sbf_i(t)$ is the SBF of the resource available to T_i in the absence of shaping, then the effective resource available to T_i in the presence of a shaper σ_i for T_i is bounded by an SBF $sbf_i^{gs} = (E_i\sigma_i) \otimes sbf_i$.

The proof is straightforward (based on results in [3] (Section 1.5.2)) and has been omitted due to space limitations.

Definitions. Next, we formally define the concepts of feasibility and optimality for greedy shapers. Let $\mathcal{C}$ be the original system without shaping, and let $\mathcal{C}_{\sigma_i}$ be the same system with the shaper σ_i applied to T_i for all $1 \le i \le N$ (recall that N is the number of tasks in the system). Further, let $\mathcal{R}(T_j, \mathcal{C})$ denote the maximum response time of task T_j in $\mathcal{C}$ for all $1 \le j \le N$.

Definition 1: A function σ_i is a *feasible shaping function* for T_i if (i) it is a good function[1], and (ii) T_i is schedulable in $\mathcal{C}_{\sigma_i}$ if T_i is schedulable in $\mathcal{C}$.

Thus, a feasible shaper always preserves the schedulability of its task. Any shaping function larger than or equal to the arrival function of a task is trivially feasible but has no effect on the task's ready pattern (i.e., as if there were no shaper), so it is not useful for our purposes.

Definition 2: σ_i is an optimal shaper of T_i iff (i) it is a feasible shaper of T_i, and (ii) for any feasible shaping function $\widehat{\sigma}_i$ of T_i, $\mathcal{R}(T_j, \mathcal{C}_{\sigma_i}) \le \mathcal{R}(T_j, \mathcal{C}_{\widehat{\sigma}_i})$ for every $T_j \in LP(T_i)$.

It implies that if a lower priority task of T_i is schedulable under a feasible shaper $\widehat{\sigma}_i$, then it is also schedulable under an optimal shaper σ_i. Thus, an optimal shaper of a task not only maximizes the number of schedulable tasks but also minimizes the maximum response times of its lower priority tasks.

When using a feasible shaper, the effective RBF of the ready jobs (output from the shaper) can be computed as follows:

[1]Since any function can be refined into a good function (e.g, by taking the sub-additive closure), we require this condition for ease of presentation.

Lemma 4.2: The effective RBF of (the ready jobs of) T_i with arrival function α_i and feasible shaper σ_i^{gs} is given by $rbf_i^{gs} = E_i(\alpha_i \otimes \sigma_i^{gs})$.

Proof: By substituting the shaping function into Theorem 4.1(1), we obtain the arrival function of the output jobs from the shaper σ_i^{gs} (i.e., input jobs to the ready queue Q_i), which is given by $\alpha_i^{gs} = \alpha_i \otimes \sigma_i^{gs}$. Hence, the effective RBF of the ready jobs of T_i is $rbf_i^{gs} = E_i \cdot \alpha_i^{gs} = E_i(\alpha_i \otimes \sigma_i^{gs})$. ■

The next lemma states the monotonicity between shapers and their resulting maximum response times of tasks:

Lemma 4.3: Suppose σ_i and $\widehat{\sigma}_i$ are two feasible shapers for T_i. Then, the following holds:

1) If $\sigma_i \le \widehat{\sigma}_i$, then $\mathcal{R}(T_j, \mathcal{C}_{\sigma_i}) \le \mathcal{R}(T_j, \mathcal{C}_{\widehat{\sigma}_i})$ for all T_j with lower priority than T_i.
2) σ_i is optimal iff $\sigma_i \le \widehat{\sigma}_i$ for all possible values of $\widehat{\sigma}_i$.

Proof: Let rbf_i^{gs} and $\widehat{rbf}_i^{gs}$ be the effective RBFs of the ready jobs of T_i when using σ_i and $\widehat{\sigma}_i$, respectively. Due to Lemma 4.2, $rbf_i^{gs} = E_i(\alpha_i \otimes \sigma_i^{gs}) \le E_i(\alpha_i \otimes \widehat{\sigma}_i^{gs}) = \widehat{rbf}_i^{gs}$. As a result, the maximum resource interference that T_i imposes on its lower-priority tasks when using σ_i is always less than or equal to that when using $\widehat{\sigma}_i$. Since the resource interference that T_i's higher-priority tasks impose on T_i's lower-priority tasks does not depend on the shaper that is used for T_i, we can conclude that $\mathcal{R}(T_j, \mathcal{C}_{\sigma_i}) \le \mathcal{R}(T_j, \mathcal{C}_{\widehat{\sigma}_i})$ for all T_j with a lower priority than T_i, which implies (1). Part (2) follows directly from part (1) and the definition of optimal shaper. ■

Since the arrival functions of the tasks are also feasible shapers, it follows from Lemma 4.3 that a system in which high-priority tasks implement feasible shaping functions that are smaller than their respective arrival functions not only preserves the schedulability of the system but also reduces the maximum response times, and hence the schedulability, of lower-priority tasks. Thus, from a schedulability and resource-demand perspective, a system with feasible shapers is always as good as, if not better than, one without shapers.

Computing the optimal shaper. Let f be an arrival function and g be the service function of a task, i.e., $g(t)$ gives the minimum number of jobs that can be completed over any interval of length t. Then, from [3], the maximum response time (delay) of the task is the maximum horizontal distance between f and g, denoted by $\mathsf{dist}(f,g)$, where $\mathsf{dist}(f,g) = \sup_{t \ge 0}\{\inf_{d \ge 0}\{d \mid f(t) \le g(t+d)\}\}$. The next theorem gives the necessary and sufficient condition for feasible shapers.

Theorem 4.4: Let $\alpha_i(t)$ and D_i be the arrival function and relative deadline of T_i, respectively, and let

$$\gamma_i(t) = \max\{0, \alpha_i(t - D_i)\}, \; \forall\, t \ge 0.$$

Then $\sigma_i \in \mathcal{F}$ is a feasible shaping function for T_i iff it is a good function and $\sigma_i \ge \gamma_i$.

Proof: ($\Rightarrow$) Let $sbf_i(t)$ be the SBF that bounds the resource available to a task T_i in $\mathcal{C}$ and let $\beta_i = \lfloor sbf_i / E_i \rfloor$. Then, $\beta_i(t)$ is the service function of T_i in $\mathcal{C}$. Since the set of tasks with higher priority than T_i in $\mathcal{C}$ and $\mathcal{C}_{\sigma_i}$ are the same, $\beta_i(t)$ is also the service function of T_i in $\mathcal{C}_{\sigma_i}$.

Suppose T_i is schedulable in $\mathcal{C}$. Then, $sbf_i(t) \ge dbf_i(t) = E_i\gamma_i(t)$, which implies $\beta_i(t) \ge \gamma_i(t)$, for all $t \ge 0$ (since $\gamma_i(t) \in$

$\mathbb{N}$). Let $a(t)$ be a release pattern of T_i, i.e., $a(t)$ denotes the number of jobs that are released over $(0,t]$ for all $t \geq 0$.

If none of the jobs in the release pattern $a(t)$ is delayed by the shaper σ_i, then all jobs of T_i also meet their deadlines in $\mathcal{C}_{\sigma_i}$. Otherwise, there exists a job J_k of T_i that is delayed by the shaper σ_i. Let t_k and t_k^s be the release time and the ready time of J_k, respectively. Figure 4 illustrates the release and ready patterns with respect to α_i, σ_i and γ_i.

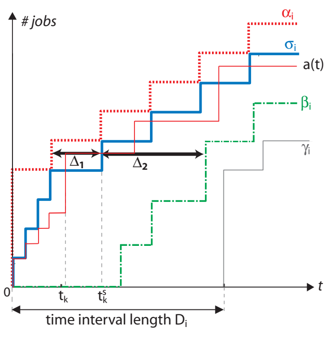


Fig. 4: Job delay under shaping.

Since σ_i is a good function and the shaper outputs J_k as soon as the resulting output pattern does not violate the arrival constraint of the shaping function, the delay of J_k at the shaper is the minimum time interval d_1 such that $a(t_k) \leq \sigma_i(t_k^s)$, with $t_k^s = t_k + d_1$. Since $a(t_k) \leq \alpha_i(t_k)$, we imply $d_1 \leq \Delta_1$, where Δ_1 is the horizontal distance between the arrival function α_i and the point $\sigma_i(t_k^s)$ (see Figure 4). Further, the maximum delay J_k experiences at the processor is the minimum interval $\Delta_2 \geq 0$ such that the number of jobs that can be completed by the processor over an interval of length $t_k^s + \Delta_2$ is at least $\sigma_i(t_k^s)$, i.e., it is the minimum time interval Δ_2 such that $\beta_i(t_k^s + \Delta_2) \geq \sigma_i(t_k^s)$. This time interval Δ_2 is indicated by the horizontal distance between $\sigma_i(t_k^s)$ and the service function β_i in Figure 4.

From the above, the maximum response time of J_k is $d_1 + \Delta_2 \leq \Delta_1 + \Delta_2 \overset{\text{def}}{=} \Delta$. There are two cases:

If $\beta_i(t_k^s + \Delta_2) \geq \sigma_i(t_k^s)$**:** Then, $\Delta_2 = 0$ and $\Delta \leq \Delta_1 \leq \mathsf{dist}(\alpha_i, \sigma_i)$. Since $\sigma_i \geq \gamma_i$, we imply $\Delta \leq \mathsf{dist}(\alpha_i, \gamma_i)$.

Otherwise: Then, Δ is no more than the horizontal distance between the arrival function α_i and $\beta_i(t_k^s + \Delta_2)$. Thus, $\Delta \leq \mathsf{dist}(\alpha_i, \beta_i) \leq \mathsf{dist}(\alpha_i, \gamma_i)$.

Hence, $\Delta \leq \mathsf{dist}(\alpha_i, \gamma_i)$. Recall that $\gamma_i(t) = \max\{0, \alpha(t - D_i)\}$ for all $t \geq 0$. This implies $\mathsf{dist}(\alpha_i, \gamma_i) = D_i$. In other words, the response time of J_k is no more than D_i. Hence, T_i is also schedulable in $\mathcal{C}_{\sigma_i}$. Because σ_i is a good function, we then imply that it is a feasible shaping function for T_i.

($\Leftarrow$) Reversely, suppose $\sigma_i(t_k) < \gamma_i(t_k)$ for some $t_k \geq 0$. Since $\gamma_i(t) = 0$ for all $t \leq D_i$ and $\sigma_i(t) \geq 0$ for all $t \geq 0$, we have $t_k > D_i$. Let $a(t)$ be the release pattern that coincides the arrival function, i.e., $a(t) = \alpha_i(t)$ for all $t \geq 0$, where $a(t)$ is the number of jobs that are released over $(0,t]$. Then, $a(t_k - D_i) = \alpha_i(t_k - D_i) = \gamma_i(t_k) > \sigma_i(t_k)$. This implies that there exists $t_k' < t_k - D_i$ such that $a(t_k') = \sigma_i(t_k)$, and subsequently, there is a job in $a(t)$ that is released at time t_k' and ready at time t_k. This job has a delay of at least $t_k - t_k'$, which is larger than D_i and thus, it misses its deadline. In other words, σ_i is not a feasible shaping function for T_i. ■

The next two corollaries follow directly from the above theorem and Lemma 4.3(2) (see also Section II for the property of the self-deconvolution).

Corollary 4.5: Let $f \in \mathcal{F}$ be such that $f(t) = \lceil B_i \cdot t / D_i \rceil$ for all $0 \leq t \leq D_i$ and $f(t) = \alpha_i(t - D_i)$ otherwise, where $B_i = \alpha_i(0 + \varepsilon)$ for an infinitely small positive value of ε. Then, $f \oslash f$ is a feasible shaping function for T_i.

Corollary 4.6: The smallest good function σ_i that is larger than or equal to γ_i is an optimal shaping function for T_i, where γ_i is defined in Theorem 4.4.

Theorem 4.7: Let $B_i = \lceil J_i / P_i \rceil$ and $\Delta_i = \min(J_i, D_i)$. Define $\sigma_i^{gs} : \mathbb{R}_+ \to \mathbb{N}$ as follows:

$$\forall t \in \mathbb{R}_+,\ \sigma_i^{gs}(t) = \begin{cases} \left\lceil \frac{B_i}{\Delta_i} t \right\rceil, & \text{if } 0 \leq t \leq \Delta_i \\ \left\lceil \frac{t + J_i - \Delta_i}{P_i} \right\rceil, & \text{otherwise.} \end{cases}$$

Then, σ_i^{gs} is an optimal shaping function for the periodic-with-jitter task $T_i = (P_i, J_i, E_i, D_i)$.

Proof: Recall that the arrival function of $T_i = (P_i, J_i, E_i, D_i)$ is given by $\alpha_i \in \mathcal{F}$ where $\alpha_i(0) = 0$ and $\alpha_i(t) = \lceil (t + J_i)/P_i \rceil$ for all $t > 0$. Let $\gamma_i(t) = \max\{0, \alpha_i(t - D_i)\}$, $\forall t \geq 0$. Then, $\gamma_i(t) = 0$ if $0 \leq t \leq D_i$, and $\gamma_i(t) = \left\lceil \frac{t + J_i - D_i}{P_i} \right\rceil$ otherwise.

The proof can be established by showing that σ_i^{gs} is the smallest good function that is larger than or equal to γ_i. Intuitively, the second part of σ_i^{gs} has the same long-term average slope as that of γ_i, whereas the slope of the first part must be no less than both B_i / J_i and B_i / D_i to ensure that the function is sub-additive. We also require that the function values are integers. Due to space constraints, we omit the details here. The claim then follows via Corollary 4.6. ■

Figure 5 shows an example that applies the above theorem.

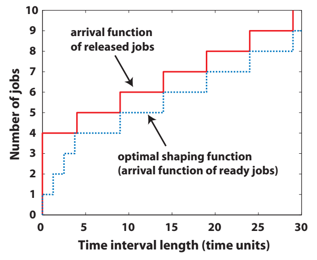


Fig. 5: Optimal shaper for a task $T = (5, 16, 1.4, 5)$, computed by Theorem 4.7. Note the reduction in the maximum bursts: in a unit time interval, there can be up 4 new jobs that are released but only one new job that will be ready for execution.

V. Schedulability Analysis with Optimal Shaping

Consider again the same task and its optimal shaper shown in Figure 5. As a result of shaping, the corresponding arrival function of the ready jobs after shaping is smaller than the original arrival function of the released jobs before shaping. This implies that the effective RBF of the ready jobs is smaller than the RBF of the released jobs. Lower RBF values mean that the task causes less interference to lower-priority tasks, which we now exploit.

We denote by $\mathcal{C}^*$ the system with N tasks $T_i = (P_i, J_i, E_i, D_i)$, where each T_i uses a feasible shaper σ_i^{gs} (e.g., the optimal shaper defined in Theorem 4.7). Recall that $rbf_i(t)$ is the original RBF of T_i and rbf_i^{gs} is the effective RBF of the ready jobs of T_i in the presence of shaping (c.f. Lemma 4.2).

Theorem 5.1: Suppose $D_i \leq P_i$ for all $1 \leq i \leq N$, and let

$$rbf_{i,\mathcal{C}^*} = rbf_i + \sum_{T_j \in HP(T_i)} rbf_j^{gs}.$$

Then, $\mathcal{C}^*$ is schedulable iff for every T_i, there exists $t_i \in [0, D_i]$ such that $rbf_{i,\mathcal{C}^*}(t_i) \le sbf(t_i)$ where $sbf(t)$ is the SBF of the processor's resource available to $\mathcal{C}^*$.

Proof: One can easily prove that a critical release pattern of T_i (i.e., a release pattern that generates the maximum demand from all its higher-priority tasks) in $\mathcal{C}^*$ happens when each higher-priority task of T_i is ready simultaneously with T_i, and all future jobs of these higher-priority tasks are ready as soon as possible. Since only the ready jobs of higher-priority tasks of T_i can interfere with the execution of T_i, for any interval of length t, the maximum resource interference of a higher priority task T_j on T_i in the system $\mathcal{C}^*$ is $rbf_j^{gs}(t)$. Further, $rbf_i(t)$ is the maximum number of execution units requested by T_i (note that σ_i does not change the requests of T_i that must be completed to meet its deadline). Hence, $rbf_{i,\mathcal{C}^*}(t) = rbf_i(t) + \sum_{T_j \in HP(T_i)} rbf_j^{gs}(t)$ is the worst-case cumulative resource request of T_i over any interval of length t. Therefore, the worst-case response time of T_i in $\mathcal{C}^*$ is the smallest time interval length t_i such that $rbf_{i,\mathcal{C}^*}(t_i) \le sbf(t_i)$. Hence, T_i meets its deadline iff $t_i \le D_i$. ■

It can be observed from the above schedulability condition that when using a shaping function larger than or equal to the arrival function of the task, the schedulability analysis of the system with shaping is reduced to the schedulability analysis of the original system. The next example illustrates the schedulability improvement when using optimal shapers.

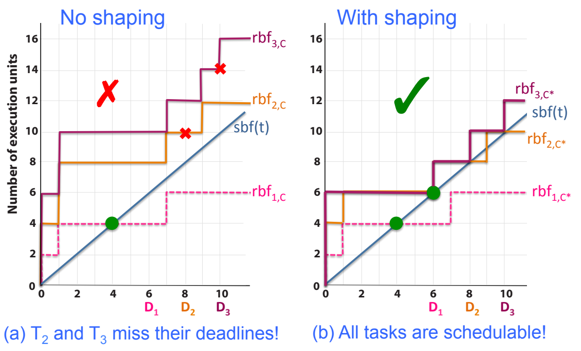


Fig. 6: Schedulability of the system in Example 2.

Example 2: Consider a system $\mathcal{C}$ with three tasks $T_1 = (6,5,2,6)$, $T_2 = (8,7,2,8)$ and $T_3 = (10,0,2,10)$, which are scheduled on a fully available unit-speed processor. T_1 has the highest priority and T_3 has the lowest priority. The cumulative RBF of T_i in $\mathcal{C}$ is denoted by $rbf_{i,\mathcal{C}}(t)$, and the SBF of the processor is denoted by $sbf(t)$. These functions are given in Figure 6(a). In this figure, the smallest time interval t at which $sbf(t) \ge rbf_{i,\mathcal{C}}(t)$ is the maximum response time of T_i when there is no shaping. From the figure, one can observe that both $rbf_{2,\mathcal{C}}(t)$ and $rbf_{3,\mathcal{C}}(t)$ meet $sbf(t)$ after the deadlines of T_2 and T_3, respectively. Therefore, T_2 and T_3 miss their deadlines.

On the other hand, when each task T_i implements its optimal shaping function (computed by Theorem 4.7), all tasks meet their deadlines. This is illustrated by Figure 6(b). As before, $rbf_{i,\mathcal{C}^*}(t)$ is the cumulative RBF of T_i in the shaping system $\mathcal{C}^*$. From the figure, the response times of T_1, T_2 and T_3 are 4, 6 and 6, respectively. Thus, all tasks are schedulable when using shaping.

VI. Integrating Shaping into Compositional Scheduling Frameworks

In this section, we demonstrate how shapers can be integrated into a compositional scheduling framework (e.g., [8], [15]). In this setting, the system is partitioned into a tree of components that are scheduled hierarchically as illustrated in Figure 7(a). There are two types of components: a *composite* component consists of a set of child components or tasks (e.g., component $\mathcal{C}$), and an *elementary* component consists of a finite set of tasks (e.g., $\mathcal{C}1$, $\mathcal{C}2$). Each component has its own local scheduler for its child components/tasks.

The schedulability analysis of the system is done by means of resource *interfaces*. An interface of a component captures the resource supply required to feasibly schedule its child tasks or child components. The interface is computed based on the resource demands of the child tasks, and/or the interfaces of any subcomponents. The interface of a component is implemented as a task, which is scheduled by the parent component's scheduler (e.g., $\mathrm{T}_{\mathcal{C}1}$). A component is schedulable if its interface is schedulable by the parent's scheduler. An interface of a component is *optimal* (resp. *bandwidth-optimal*) if its SBF (resp. its bandwidth) is smaller than, or equal to, that of any other feasible interface of the component.

We illustrate our method using the explicit deadline periodic (EDP) interface [8], since it is efficient and can easily be transformed into a task. An EDP interface is characterized by a period Π, an execution time Θ, and a deadline Δ. It represents a resource that supplies Θ execution time units during each period of length Π, within the first Δ time units after the start of each period. It has a bandwidth of Θ/Π, and it can be transformed into a periodic task with period Π, WCET Θ and deadline Δ. The SBF of an EDP interface is given in [8].

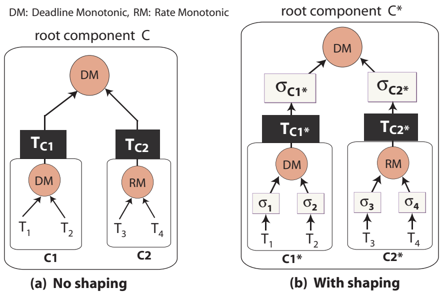


Fig. 7: A two-level compositional scheduling system.

Integration of shapers: We apply shaping to every component that uses a FP local scheduler (e.g., RM, DM in the above example). Each task T_i within any such component $\mathcal{C}$ implements its own shaper (i.e., its released jobs will first pass through the shaper before they are scheduled by $\mathcal{C}$'s scheduler). Figure 7(b) shows the system with shaping that corresponds to the original one in Figure 7(a). We refer to $\mathcal{C}$'s counterpart (with feasible shaping added) as $\mathcal{C}^*$. If T_i is a periodic task with jitter, its shaping function is computed by Theorem 4.7. The shaping function of any other general task model is computed based on the task's arrival function by using Corollary 4.5.

In Section V, we have shown that, if a system is schedulable under FP, then the same system with feasible shapers is also schedulable. Since this claim was proven for any general resource characterized by an SBF, the same result holds for a component within the compositional scheduling framework. That is, if $\mathcal{C}$ is schedulable under FP, then $\mathcal{C}^*$ is also schedulable under FP. Hence, the compositional schedulability analysis in presence of shaping can be done in the same manner as the conventional setting, except that the interface of a component with shaper is computed based on the schedulability condition stated in Theorem 5.1. Next, we elaborate this interface computation for any given $\mathcal{C}^*$.

Interface computation for a component with shaping: Let σ_i^{gs} be the shaper of T_i as described above, for all T_i in $\mathcal{C}^*$. As usual, D_i denotes the deadline of T_i, $rbf_i(t)$ denotes the RBF of T_i (c.f. Equation (2)), and rbf_i^{gs} denotes the effective RBF of the ready jobs of T_i in presence of shaping (c.f. Lemma 4.2).

Theorem 6.1: An interface $\mathcal{I}$ can feasibly schedule $\mathcal{C}^*$ iff

$$\forall\, T_i \in \mathcal{C}^*,\ \exists\, t_i \in [0, D_i]:\ \ rbf_{i,\mathcal{C}^*}(t_i) \le sbf_{\mathcal{I}}(t_i)$$

where $rbf_{i,\mathcal{C}^*} = rbf_i + \sum_{T_j \in HP(T_i)} rbf_j^{gs}$ is the cumulative RBF of T_i in $\mathcal{C}^*$ (which accounts for interference of its higher priority tasks), and $sbf_{\mathcal{I}}(t)$ is the SBF of $\mathcal{I}$.

Proof: The theorem holds due to the schedulability condition of components with shaping given in Theorem 5.1. ■

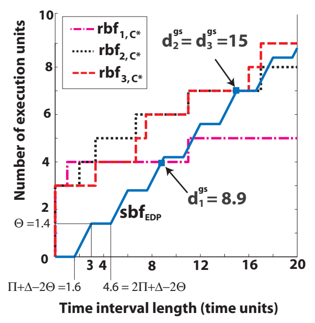
Corollary 6.2: An interface $\mathcal{I}_{\text{opt}}$ is optimal (bandwidth-optimal) for $\mathcal{C}$ iff it has the minimum SBF (minimum bandwidth) among that of all interfaces $\mathcal{I}$ that satisfy the schedulability condition given by Theorem 6.1.

Based on Corollary 6.2, we can compute the optimal (bandwidth-optimal) interface for any shaping component C^* by following the same procedure that was used for the component $\mathcal{C}$ without shaping; the only difference is that we use the updated cumulative RBFs $rbf_{i,\mathcal{C}^*}$ instead of $rbf_{i,\mathcal{C}}$. The computation of interfaces of components that do not use shapers (i.e., non FP components), and the transformation from interfaces to tasks, can be done exactly as in the conventional case (see [8] for EDP interfaces).

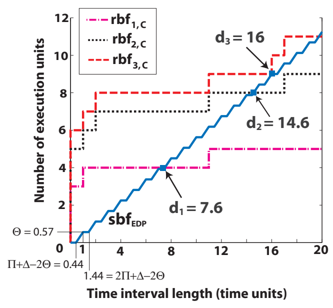
Since $rbf_{i,\mathcal{C}^*} \le rbf_{i,\mathcal{C}}$ for all FP components, the resource needed to schedule the components when using shapers is always smaller than or equal to the resource required in the conventional setting. We end this section with an example to illustrate the effect of shaping on its interface.

Example 3: Consider an elementary FP component $\mathcal{C}$ that consists of three tasks $T_i = (P_i, J_i, E_i, D_i)$, where $T_1 = (10, 29, 1, 10)$, $T_2 = (15, 28, 1, 15)$, $T_3 = (16, 0, 1, 16)$, and the priority order is $T_1 > T_2 > T_3$. Our goal is to compute the minimum-bandwidth EDP interface for this component, where the maximum interface period is chosen to be $\Pi_{\max} = 32$.

We first compute the shaper for each task using Theorem 4.7, and derive the corresponding cumulative RBF $rbf_{i,\mathcal{C}^*}$ for each task T_i (c.f. the step functions in Figure 8(a)). We then vary the interface period Π from 1 to $\Pi_{\max}$, and for each value Π, we find the corresponding Θ and Δ that result in a minimum bandwidth interface (Π, Θ, Δ) whose SBF satisfies Theorem 6.1. The minimum-bandwidth interface among the interfaces for all values of Π is given by $(\Pi = 3, \Theta = 1.4, \Delta = 1.4)$, which has a bandwidth of $0.4667 (= 1.4/3)$. One can



(a) Shaping: $BW = 0.4667$, $(\Pi, \Theta, \Delta) = (3, 1.4, 1.4)$.



(b) No shaping: $BW = 0.56$, $(\Pi, \Theta, \Delta) = (1, 0.56, 0.56)$.

Fig. 8: Minimum-bandwidth EDP interface for Example 3.

validate based on Figure 8(a) that this interface is sufficient to schedule the task set because the maximum response time d_i^{gs} of T_i is always no more than D_i.

In contrast, as illustrated in Figure 8(b), the minimum interface of the component in the absence of shaping is $(\Pi = 1, \Theta = 0.56, \Delta = 0.56)$, which has a bandwidth equal to 0.56. Hence, shaping reduces the interface bandwidth by more than 16.6% in this single component alone.

Remarks: Since a feasible shaper for a task can be computed based solely on its arrival function (Corollary 4.5), our technique is applicable to any task model that can be characterized by an arrival function (e.g., a sporadic task). Further, it can be plugged into any compositional scheduling framework that uses supply/demand analysis for interface computation.

VII. Shapers in mixed-criticality systems

In this section, we present an application of shapers to mixed-criticality systems. In such systems, each task has – in addition to its conventional timing parameters – a criticality level. There are two scheduling objectives: (1) to guarantee that all high-criticality (HC) tasks meet their deadlines; and (2) if objective (1) is feasible, to minimize the deadline miss ratios of low-criticality (LC) tasks, in decreasing order of priority. Note that (2) is only one possible objective, which we follow so as to confirm to the FP scheduler as much as is possible. In general, the criticality of a task can change at runtime, depending on the operating mode of the system. As a first step, we present the technique for a single-mode scenario and for two levels of criticality. Our goal is to illustrate that shaping is a promising way to achieve the mixed-criticality scheduling objectives.

In the setting we consider, a shaper is feasible for a task T_i iff it preserves the schedulability of T_i's lower-priority HC tasks. A shaper is optimal for T_i iff it is the smallest feasible shaper for T_i that preserves the schedulability of T_i. We require shapers are good functions.

Basic idea: We will use a combination of shaping and interface techniques. By appropriately shaping LC tasks, we can ensure that HC tasks meet their deadlines – even when they have low priorities – while reducing the deadline miss ratios of the LC tasks. Intuitively, since shapers can control the number of tasks that are ready, we can reduce the resource contention on HC tasks by designing shapers with sufficiently small shaping functions for LC tasks.[2] At the same time, by choosing these functions as large as possible while still ensuring schedulability of HC tasks, we can increase the resource utilization and minimize the LC tasks' deadline miss ratios. Towards this, we compute for each task T_i an interface, called the *critical interface*, that captures the minimum resource supply needed to feasibly schedule all HC tasks with a priority smaller than or equal to T_i's. We then choose a shaping function for each LC task such that the remaining resource after processing the task satisfies its critical interface.

From the scheduling perspective, this approach offers two main benefits: (1) it does not modify the scheduler, and thus enables the conventional FP algorithm to be used in mixed-criticality systems without violating the timing guarantees of HC tasks; and (2) since the reduction in resource contention is achieved entirely through shaping (and not by modifying the scheduler), this approach can potentially be combined with other mixed-critically scheduling algorithms to further improve schedulability.

Computing shapers for mixed-criticality tasks: Since the goal of the shaper for an HC task is the same as in systems without mixed criticality (i.e., to ensure the task's schedulability and to minimize the resource interference on lower priority tasks), the optimal shaper for HC tasks can be computed using Theorem 4.7. The same theorem can be used for LC tasks with a priority below that of all HC tasks, since such tasks cannot interfere with any HC tasks. Let τ_{LCHP} be the set of remaining tasks, i.e., LC tasks whose priority exceeds that of at least one HC task. Let T_i be a task in τ_{LCHP}. Then T_i's optimal shaper can be derived from its critical interface and the SBF of the resource available to it. The critical interface of a task is defined by Lemma 7.1, which holds due to Corollary 6.2:

Lemma 7.1: Let $\mathcal{C}_i$ be a virtual component that consists of all HC tasks that have a priority lower than or equal to T_i's, such that each task in $\mathcal{C}_i$ implements its own optimal shaper as described above. Then the critical interface $\mathcal{I}_i$ of T_i is the minimum SBF that satisfies the schedulability condition of $\mathcal{C}_i$, which is given by Theorem 6.1 (Section VI).

From Lemma 7.1, σ_i is a feasible shaper for T_i iff the SBF of the remaining resource after processing the output jobs of σ_i is at least $\mathcal{I}_i$. Let β_i be the SBF of the minimum resource available to T_i. The effective RBF of the output jobs of σ_i is $rbf_i^{gs} = E_i(\alpha_i \otimes \sigma_i)$, where α_i and E_i are the arrival function and WCET of T_i (see Lemma 4.2). Combine with the results in Section II, σ_i is feasible for T_i iff $\mathsf{Remain}(\beta_i, rbf_i^{gs}) \geq \mathcal{I}_i$ (c.f. Equation (3)). Since α_i and σ_i are good function, the maximum solution for $\alpha_i \otimes \sigma_i$ is $\sigma_i^{\mathsf{crt}} \stackrel{\text{def}}{=} \mathsf{closure}\big(\lfloor \frac{\mathsf{RTInverse}(\mathcal{I}_i,\beta_i)}{E_i} \rfloor\big)$ where $\mathsf{closure}(f)$ is the sub-additive closure of f (c.f. Equation 4).

Further, let σ_i^{gs} be the shaper computed using Theorem 4.7. Recall that σ_i^{gs} is the smallest shaping function that ensures that T_i meets its deadline. Also, $\sigma_i^{gs} \leq \alpha_i$. As a result, the optimal shaping function for T_i is $\min\{\sigma_i^{\mathsf{crt}}, \sigma_i^{gs}\}$.

Based on the above results, we can compute the optimal shapers for the tasks in a mixed-criticality system in decreasing order of priority based on Theorem 7.2. In this theorem, $\beta_i(t)$ denotes the SBF of the resource available to T_i, and $\beta_0(t)$ is the SBF of the total resource given by the processor.

Theorem 7.2: Suppose that, for all $1 \leq i, j \leq N$, where N is the number of tasks in the system, T_i has higher priority than T_j if $i < j$. Let α_i and E_i be the arrival function and the WCET of T_i, respectively. Then the optimal shaper for T_i is $\sigma_i = \sigma_i^{gs}$ if $T_i \notin \tau_{LCHP}$; otherwise, $\sigma_i = \min\big\{\sigma_i^{gs}, \mathsf{closure}\big(\lfloor \frac{\mathsf{RTInverse}(\mathcal{I}_i,\beta_i)}{E_i} \rfloor\big)\big\}$,where $\beta_1 = \beta_0$ and $\beta_i = \mathsf{Remain}(\beta_{i-1}, rbf_{i-1}^{gs})$ for all $2 \leq i \leq N$.

It can be derived that all the HC tasks are schedulable iff $\beta_0 \geq \mathcal{I}_1$, and an LC task is schedulable iff its shaping function satisfies the feasibility condition in Theorem 4.4 (Section IV). Since the system behaves as in the setting without mixed-criticality after the shapers are fixed, the maximum response time of a task can be computed as was done in Section V.

VIII. Evaluation

In this section, we evaluate the proposed technique on randomly generated fixed-priority task systems, in both standard FP scheduling and compositional scheduling settings. In the fomer, we quantify (i) the degree to which shaping can improve the schedulability of low-priority tasks under high system load situations, and (ii) the relationship between the tasks' jitter values and the schedulability improvement achieved by shaping. In the later, we quantify the amount of resource interface bandwidth that can be saved by using shapers in the FP components.

Experimental setup: We implemented the analysis techniques from Sections V and VI in Matlab by extending the RTC Toolbox [22]. For each scheduling setting, we randomly generated a collection of periodic-with-jitter task sets of size N (specified below); and each task's deadline was equal to its period. The following parameters were chosen uniformly at random: the task periods (as integers from $[100, 1000]$), the task set's total utilization (from $[U_{\min}, U_{\max}]$), the utilization values of the individual tasks (from $(0,1)$, scaled by the generated utilization of the task set) and each task's jitter-to-period ratio (from $(0,2)$). The tasks' priorities were assigned according to the Deadline Monotonic algorithm; ties were broken based on the task indices.

A. Schedulability evaluation

For this experiment, we generated 1000 task sets with $N = 200$ tasks per set. Since we were interested in the schedulability of the system, which is critical under high system load situations for FP, we chose the utilization of each task set between $U_{\min} = 0.7$ and $U_{\max} = 0.9$. For each generated task set, we computed the number of tasks that missed their deadlines, based on their maximum response times.

Figure 9 shows the number of unschedulable tasks (i.e., deadline misses) for each task set with and without shaping. The task sets are sorted by increasing utilization; the filled and empty circles correspond to the results for individual task sets with and without shaping, respectively. The results

[2]A smaller shaping function allows fewer jobs to arrive at the ready queue over any time interval of a given length.

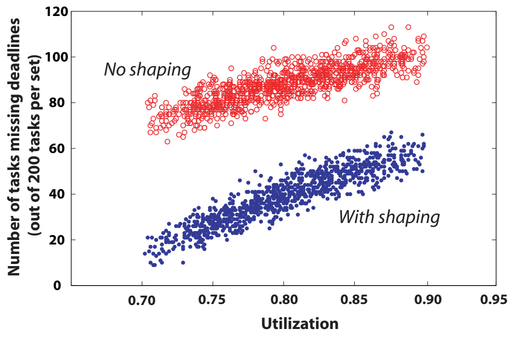


Fig. 9: Total number of deadline misses for each task set.

show that the number of deadline misses without shaping was strictly above the number of deadline misses with shaping, consistently across all utilization values. For example, at a utilization of 0.8, approximately 90 (out of 200) tasks miss their deadlines without shaping, but only 40 tasks miss their deadlines with shaping. In other words, shaping reduced the number of deadline misses by 2.25 times. Note also that the average reduction for the other utilization values is similar.

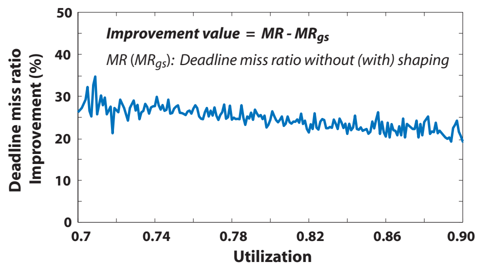


Fig. 10: Effect of shaping on reducing deadline miss ratio.

Figure 10 further illustrates the improvements when using shaping with respect to deadline miss ratio (i.e., the ratio of the number of unschedulable tasks to the total number of tasks). Each improvement value is defined as $\mathsf{MR} - \mathsf{MR}_{gs}$, where MR_{gs} (MR) is the deadline miss ratio with (without) shaping. The results show that shaping can effectively reduce the deadline miss ratio of the system by 20% to 35%.

Under high load situations, we also observe that, as the utilization increases, the reduction achieved by shaping tends to decrease slightly. This is expected: intuitively, if we keep the same number of tasks, an increase in the task set's utilization implies an increase in the tasks' WCETs (or a reduction in the tasks' periods). Hence, the tasks that used to be schedulable in presence of shaping but unschedulable without shaping now also become unschedulable under shaping, due to their higher WCETs (or reduced periods).

Since shaping is only effective when tasks experience jitter, we next investigated the relationship between jitter and the performance of shaping. We varied the jitter-to-period ratio from 0 to 2, in increments of 0.01. For each ratio, we generated 50 task sets, with $N = 100$ tasks per set and a utilization of 0.9. The other parameters were chosen as before.

Figure 11 shows the average improvement for each jitter-to-period ratio. Here, the improvement starts at 0 and increases as the jitter-to-period ratio increases. This trend continues until the improvement reaches a peak (when the jitter is approximately equal to the period), and it slowly decreases again after that. This near-bell-shape behavior is expected because when there is no jitter, the system with shaping behaves exactly as the one without shaping, so there is no improvement.

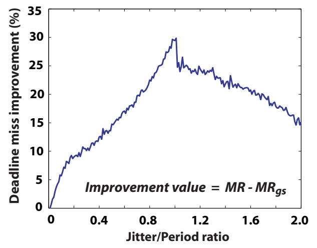


Fig. 11: Correlation between deadline miss ratio reduction and jitter.

As the jitter values increase, more and more tasks become unschedulable without shaping but remain schedulable under shaping, so the improvement increases. As was discussed in the introduction, such large jitters happen in many real-time systems, such as distributed automotive networks. Observe also that, as is expected, when the jitter is too large, the task set becomes heavily overloaded, so more and more tasks become unschedulable even with shaping.

B. *Interface computation evaluation*

To illustrate the effect of shaping on component interfaces, we considered the interface computation of a FP elementary component that was part of a compositional scheduling framework. For our experiment, we used the EDP resource model as the interface representation, and we applied the technique from Section VI to compute the optimal interface.

We generated 800 component workloads, with utilization values drawn uniformly at random from $[0.2, 0.9]$. Each component workload had 20 periodic tasks with jitter; the other parameters were chosen as described under experimental setup, except that the jitter-to-period ration was chosen from $(0, 1)$. For each generated component workload, we computed the optimal EDP interfaces for the component with and without shaping. The maximum interface period was set to 1000.

Figure 12 shows the optimal bandwidths of the EDP interfaces corresponding to the generated component workloads, sorted by increasing workload utilization. The corresponding cumulative distribution functions (CDFs) of the interface bandwidth are shown in Figure 13. As expected, when the workload utilization increases, the interface bandwidth also tends to increase.[3] In both figures, the interface bandwidth with shaping is always smaller than the interface bandwidth without shaping, across all component's workload utilization values. Furthermore, the number of component workloads that require an interface bandwidth larger than 1 (highlighted by the shaded area in Figure 13) without shaping is at least 20% larger than with shaping. Such component workloads are not schedulable on a unit-one processor, even it is fully available.

In general, the above results show that shapers not only reduce the interface bandwidth values but also minimize the percentage of component workloads that require larger interface bandwidths. This is illustrated more explicitly by Figure 14, which shows the improvement of interface bandwidth with shaping, compared to the conventional component

[3]Note that, the interface bandwidth in the presence of jitter is not strictly an increasing function of the workload utilization because it depends on not only the workload utilization but also on the jitter values of the tasks.

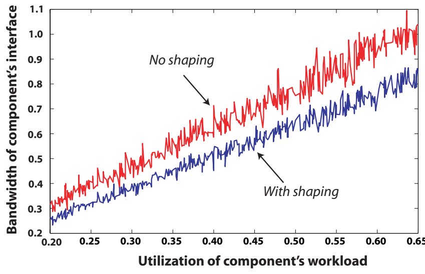


Fig. 12: Interface bandwidth with and without shaping.

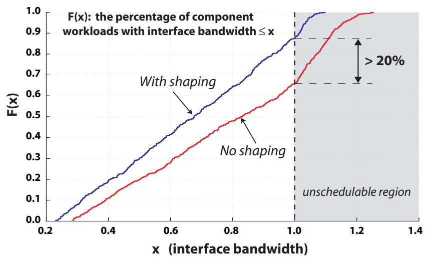


Fig. 13: CDF of the interface bandwidth with and without shaping.

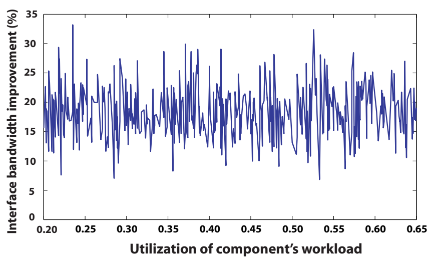


Fig. 14: Interface bandwidth improvement achieved using shaping.

without shaping. (The improvement percentage is defined by $100*(1-\mathrm{bw}^{\mathrm{gs}}/\mathrm{bw})$, where $\mathrm{bw}^{\mathrm{gs}}$ and bw are the interface bandwidths with and without shaping, respectively.) The figure shows that, by shaping the tasks within the component, we can reduce the component's interface bandwidth by up to 33%, or 17.5% on average. Moreover, the bandwidth reduction is observed across all workload utilizations. Therefore, we can conclude that shaping can minimize the resource needs (in terms of bandwidth) not only when the component is heavily loaded but also when it has a very light load. In other words, for components with periodic tasks with jitter, it is always advantageous to implement shapers.

In this evaluation, we have focused on the bandwidth reduction within a single component. Since the interface bandwidth of the system increases proportionally with the number of components, the total bandwidth that can be saved by using shaping is also accumulated as the number of components increases. This is also confirmed by our evaluation results; due to space constraints, we omit the details here.

IX. Conclusion

We have presented a shaping-based technique that reduces resource interference and improves schedulability of fixed-priority systems. We have developed a closed-form solution for computing the optimal shaper for periodic tasks with jitter and for tasks that are characterized by an arrival function. Also, we have presented a new schedulability analysis for systems with shaping. We have also shown how shapers can be integrated with compositional scheduling frameworks, and we have described an application of shapers in mixed-criticality systems. Our evaluation on random task sets shows that shaping is highly beneficial: it can help reduce the system's deadline miss ratio by up to 35%, and it can reduce the interface bandwidth up to 33%.

Shaping appears to be a promising technique to explore in the context of real-time systems, and this paper is only a first step. For instance, observe that shapers can be computed independent of the underlying resource, so it seems feasible to apply them to multiprocessor scheduling and distributed systems. This could be an interesting direction for future work. Extensive evaluation of the mixed-criticality shaping technique presented in Section VII and its extensions to multi-mode scenarios and more than two criticality levels is another promising future direction for mixed-criticality systems. We also plan to investigate the use of shaping in conjunction with more complex methods for improving resource interference, such as [24] and [17]. Finally, we plan to implement our technique in the RT-Xen virtualization platform [10] to bring its benefits to end-users.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Journal of Real-Time Systems*, 8(5):284Ð–292, 1993.

[2] Avionics Electronic Engineering Commitee(ARINC). *ARINC Specification 653-2: Avionics Application Software Standard Interface: Part 1 - required services*. Aeronautical Radio INC, Maryland, USA, 2005.

[3] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume LNCS 2050. Springer, 2001.

[4] B. B. Brandenburg, H. Leontyev, and J. H. Anderson. An overview of interrupt accounting techniques for multiprocessor real-time systems. *J. Syst. Archit.*, 57(6):638–654, June 2011.

[5] R. Bril, J. Lukkien, and W. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42:63–119, 2009.

[6] A. Burns. Preemptive priority-based scheduling: an appropriate engineering approach. In *Advances in real-time systems*, pages 225–248. Prentice-Hall, Inc., 1995.

[7] R. Dobrin and G. Fohler. Reducing the number of preemptions in fixed priority scheduling. In *ECRTS*, 2004.

[8] A. Easwaran, M. Anand, and I. Lee. Compositional Analysis Framework Using EDP Resource Models. In *RTSS*, 2007.

[9] P. Kumar and L. Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *DAC*, 2011.

[10] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *RTAS*, 2012.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan 1973.

[12] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, Inc, 2001.

[13] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *RTSS*, 2002.

[14] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60–67, Apr. 2003.

[15] I. Shin and I. Lee. Compositional Real-time Scheduling Framework with Periodic Model. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–39, 2008.

[16] J. Sun and J. Liu. Synchronization protocols in distributed real-time systems. In *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on*, pages 38–45. IEEE, 1996.

[17] A. Thekkilakattil, A. S. Pillai, R. Dobrin, and S. Punnekkat. Reducing the number of preemptions in real-time systems scheduling by cpu frequency scaling. In *RTNS*, 2010.

[18] K. W. Tindell. An extendible approach for analysing fixed priority hard real-time tasks. *J. Real-Time Systems*, 6, 1994.

[19] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance analysis of greedy shapers in real-time systems. In *DATE*, 2006.

[20] 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.

[21] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *RTAS*, 2006.

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

[23] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *RTCSA*, 1999.

[24] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *RTCSA*, 2009.

[25] A. Zuhily and A. Burns. Optimal (d-j)-monotonic priority assignment. *Information Processing Letter*, 103(6):247–250, 2007.