Journal of Circuits, Systems, and Computers
© World Scientific Publishing Company

Schedulability Analysis towards Arbitrarily Activated Tasks in Mixed-Criticality Systems

Biao Hu[†], Kai Huang[‡], Gang Chen[♯], Long Cheng[†], Dongkun Han[§] and Alois Knoll[†]

[†] *Institute of Robotics and Embedded Systems*
Technical University Munich, Boltzmannstr. 3,
Garching 85748, Germany,
{*hub, chengl, knoll*}*@in.tum.de*

[‡] *School of Data and Computer Science,*
Sun Yat-sen University, Xiaoguwei Island, Panyu District,
Guangzhou 510006, China,
huangk36@mail.sysu.edu.cn

[♯] *College of Computer Science and Engineering,*
Northeastern University, Wenhua Road, Heping District,
Shenyang 110819, China,
chengang@cse.neu.edu.cn

[§] *Department of Aerospace Engineering,*
University of Michigan, Francois-Xavier Bagnoud Building,
1320 Beal Ave, Ann Arbor, MI 48109, USA,
dongkunh@umich.edu

The integration of mixed-critical tasks into a platform is an increasingly important trend in the design of real-time systems due to its efficient resource usage. With a growing variety of activation patterns considered in real-time systems, some of them capture arbitrary activation patterns. As a consequence, the existing scheduling approaches in mixed-criticality systems, which assume the sporadic tasks with implicit deadlines, have sometimes become inapplicable or are ineffective. In this paper, we extend the sporadically activated task model to the arbitrarily activated task model in mixed-criticality systems with the preemptive fixed-task-priority schedule. By using the event arrival curve to model task activations, we present the necessary and sufficient schedulability tests that are based on the well-established results from Real-Time Calculus. We propose to use the busy-window analysis to do the sufficient test because it has been shown to be tighter than the sufficient test of using Real-Time Calculus. According to our experimental results, for sporadic task sets, our proposed test can achieve the same performance as the state-of-the-art schedulability test. However, compared with the previous schedulability analysis of preemptive fixed-task-priority, our approaches can handle more general tasks with blocking, jitter, and arbitrary deadlines.

Keywords: Schedulability Analysis; Mixed-Criticality Systems; Arrival Curve.

[‡] Corresponding author.

1. Introduction

Integrating tasks with different importance or criticality on a common computing platform has become a prevalent design paradigm in real-time and embedded systems. One challenge of designing such a *mixed-criticality system* (MCS) is how to certify the tasks based on their own criticality levels. To address this concern, an important character of the MCS is that system parameters become dependent on the criticality level of the tasks. Most paper (see a relevant survey [1]) assume that in a dual-criticality system, a high criticality task has a low level *worst case execution time* (WCET) as its low level system guarantee and a high level WCET as its high level system guarantee, while a low criticality task only has a low level WCET because a low criticality task only needs to be guaranteed in a low level. During the runtime, if any high criticality task overruns its given low level WCET, the system will abandon all low criticality tasks in order to leave enough system resource to meet the high level guarantee of high criticality tasks. The system is called *mixed-criticality schedulable* if all tasks meet their deadlines in the case all tasks execute below their low level WCETs and high criticality tasks can meet their deadlines even when they execute over their low level WCETs.

To simplify the schedulability analysis of a system following the above scheme, tasks in MCSs are often modeled as the *sporadic* tasks that only define a minimum inter-activation interval (also called period) [1,2]. Based on the sporadic task model, the state-of-the-art schedulability test in the fixed-priority scheduled system is the response time analysis proposed in [2]. In the earliest-deadline-first scheduled system, the often used schedulability tests are EDF-VD [3] and those demand-based schedulability analysis [4–6].

Although the sporadic task model simplifies the schedulability analysis and can also represent many *nondeterministic* activation patterns by assuming that the task can be activated in every period, such representation may not be effective in the schedulability analysis. For instance, a simple approach to dealing a periodic task with a jittery release pattern is to transform it into a new sporadic task with a shorter period [7]. While this approach is safe, the transformation can lead to overly pessimistic schedulability analysis results. If this shorter period is smaller than the WCET of this task, it is impossible to schedule this task by modeling it as a sporadic task, because the sporadic task is assumed to be activated in every shorter period and the deadlines deem to be missed. The real situation is that the task cannot be activated in every shorter period. This task may be schedulable in MCSs, which however will be determined as not schedulable by modelling it as a sporadic task.

The periodic task with a jittery release or a task with burst activations, often exists in many reactive embedded systems. The jitter may come from release-delay overheads induced by tick-driven scheduling [8], execution of interrupt service routines [7], or I/O overheads. The delays by scheduling and data dependencies may also cause the jitter. In ARINC avionics systems, different tasks scheduling partitions are connected over a switched Ethernet. Due to the network delay, tasks in a partition are not always released strictly periodically, but with a certain jitter [9]. In the automotive systems [10], a lot of event streams that are used to activate tasks suggest the use of more general event stream models. In the traditional real-time systems, complex activation patterns are often modeled as the *arrival curve* or the *minimum distance function* to compute the system throughput and delay by applying the Modular Performance Analysis [11] (under the framework of Real-Time Calculus [12, 13]) or the Compositional Performance Analysis [14]. In contrast with certifying the system throughput and delay on one level, the throughput and delay need to be certified based on the criticality of tasks in Mcss, which complicates the analysis.

In this paper, the schedulability of dual-criticality system with arbitrarily activated tasks is analyzed. The schedulability analysis towards the arbitrarily activated tasks in Mcss, however, is nontrivial. In contrast with that at most one carry-over job (released but not finished) exists at the time of system mode switch for each sporadic task, there will be several carry-over jobs at once for each arbitrarily activated task. The exact number of carry-over jobs is difficult to obtain because the carry-over jobs depend on the specific activation patterns of this task and all higher priority tasks. This complicates the derivation of a tight bound of the worst-case response time (Wcrt) of a task. Furthermore, for a given task set whose priorities are to be assigned, the exhaustive searching over all possibilities of priority assignment is time-consuming. Hence, a more effective approach should be used to assign priorities.

Being aware of the above, this paper proposes a necessary and two sufficient schedulability tests to extend the classical sporadic task model to the arbitrarily activated task model in Mcss with the preemptive fixed-task-priority. The detailed contributions are as follows:

- In Mcss, we extend the classic sporadic task model to the arbitrarily activated task model by presenting a necessary and two sufficient schedulability tests. We also show that Audsley's algorithm is applicable in all tests.
- By using the arrival curve to represent the upper bound of task activations, we integrate the well-established results from Real-Time Calculus to analyze the schedulability of arbitrarily activated tasks in Mcss.
- By using the minimum distance function to model task activations and using the busy-window analysis to compute the Wcrt of tasks, we present a tighter sufficient schedulability test than the schedulability test of using Real-Time Calculus framework.
- It is demonstrated that, for the sporadic tasks, the sufficient test using the busy-window analysis can achieve the same schedulability as the AMC-max test in [2]. However, compared with AMC-max, our test can handle the tasks with blocking, jitter, and arbitrary deadlines. It is verified by experiments that, the increase of jitter decreases the system schedulability,

while the increase of relative deadlines increases the system schedulability.

The remainder of this paper is structured as follows. Next section reviews related work. Section 3 presents the system model, settings and a motivation example. Section 4 provides the background theory. Section 5 presents the necessary test, and Section 6 presents two sufficient tests. Section 7 presents the experimental results and the last section concludes this paper.

2. Related Work

Since the first paper on the verification of a proposed Mcs in 2007 [15], the design and analysis towards a Mcs have ranged from the uniprocessor to the multiprocessor in many various aspects. For example, there is some research that focuses on maintaining the system fault-tolerant property [**?**, 16], and there is also some research focusing on memory architecture [17]. For more works in Mcss, we recommend the interested readers to a comprehensive review [1]. In this paper, we mainly focus on the scheduling analysis of the uniprocessor with the fixed-task-priority schedule.

In [15], Vestal proposed to use Audsley's algorithm [18] to assign the priorities in Mcss with fixed-priority policy. Audsley's algorithm was proven in [19] to be optimal for assigning priorities to sporadic tasks with different criticality levels. In Vestal's approach, the priorities of tasks with different criticalities are allowed to be interleaved, leading all tasks to be evaluated as if they were of the highest criticality. By implementing the Mcs in Ada, it was reported in [20] that higher resource usage can be achieved by monitoring task execution time and preventing execution time overruns. With such a platform that can monitor how long individual jobs have been executed, two new schemes called AMC-rtb and AMC-max that dominate Vestal's approach were proposed in [2, 21]. The AMC-rtb approach was extended to incorporate preemption thresholds in [22, 23], where a reduction in stack usage and an improved performance were demonstrated. The dual-criticality model was extended to an arbitrary number of criticality levels by using the same analyzing scheme as AMC-rtb [24]. In this paper, we take the same strategy in the scheduling as [2, 21], i.e., stop the execution of LO-critical tasks after any high criticality task overruns its low level worst-case execution time.

All the aforementioned work is based on the task model that is activated sporadically and the assumption that its Wcrt is less than the minimum activation interval. With this assumption, there is at most one carry-over job (released, but not finished) when the system criticality switch is invoked. The computation of Wcrt and the derivation of demand-bound function are often based on this carry-over job [2, 4, 21]. Without this assumption, the previous methods of computing Wcrt and deriving demand-bound function thus cannot be used.

The sporadic task model was extended to include the release jitter in [19], based on which the sensitivity analysis was presented. However, there are fundamental differences between this analysis and the one in this paper. First, the analysis in [19] does not consider the prevention of execution time overruns. Second, although with a jitter, it still assumes that tasks are activated periodically, which is not enough to represent the arbitrary activated tasks. Third, it assumes that tasks have constrained-deadlines (i.e., relative deadlines are not greater than periods), while our schedulability tests do not need such assumption. Furthermore, the detailed influence of jitter on the system schedulability is not evaluated in [19].

Despite of the unchanged period of sporadic tasks that were often modeled in MCSs, a task model in which the period differs among different criticality modes, instead of the WCET, was introduced in [25]. While this setting allows the period transformation, the sporadic task model is not changed in every mode. In [26,27], by prioritizing all LO-critical tasks over HI-critical tasks in MCSs, two new monitoring approaches were proposed to monitor the workload of LO-critical tasks at runtime. Although the proposed monitoring approaches consider the arbitrarily activated tasks, there is no mode switch in their MCSs. Thus, it is still not clear how to monitor the workload in MCSs if a mode switch exists.

3. System Model and Motivations

In this section, we present the event model, our system settings, and a motivation example to show the inadequacy of the existing approaches. The key notations described in this Section and used in the rest of this paper is summarized in Tab. 1.

3.1. *Event Model*

We consider that a task is activated by an event [11]. Task activations in the system can be expressed as an event stream. A trace of such an event stream is described by means of an 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 curve provides an abstract event stream model that represents the maximum and minimum number of events that are seen in a time interval.

Definition 1. (Arrival Curve [11]) *Denote $R[s,t)$ as the number of events that arrive on an event stream in the time interval $[s,t)$. Then, R, $\overline{\alpha}^u$ and $\overline{\alpha}^l$ represents the upper and lower bound on the number of event in any interval $t-s$, that is,*

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

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

A similar concept corresponding to the upper arrival curve is the minimum distance function [14].

Definition 2. (Minimum Distance Function [14]) *The minimum distance*

Table 1: Notations used throughout the paper

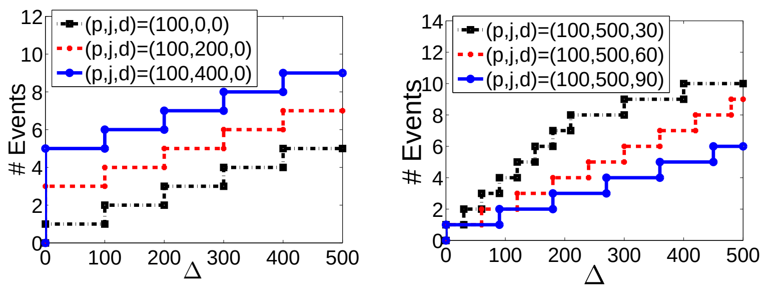
Symbol	Description
n	Number of tasks in the task set
τ	Task set
τ_i	i-th task $\in \tau$
L_i	Criticality of τ_i
D_i	Relative deadline of τ_i
$C_i(LO)$	LO mode WCET of τ_i
$C_i(HI)$	HI mode WCET of τ_i
p_i	Period of τ_i
j_i	Jitter of τ_i
d_i	Minimum distance of τ_i
$hp(i)$	Tasks whose priorities are higher than τ_i
$hpL(i)$	LO-critical tasks whose priorities are higher than τ_i
$hpH(i)$	HI-critical tasks whose priorities are higher than τ_i
$\overline{\alpha}_i^u$	Upper arrival curve of τ_i
$\delta_i(k)$	Minimum interval among any $k+1$ events of τ_i
β_i^l	Lower service curve for τ_i
β_i^{lLO}	LO mode lower service curve for τ_i
β_i^{lHI}	HI mode lower service curve for τ_i
$\mathrm{Buf}_j^{\max}$	Maximum number of events backlogged in LO mode
$B_i^{LO}(q,\delta_i)$	q-event busy window of τ_i in LO mode
s	System switches to HI mode at time s
$B_i^s(q,\delta_i)$	q-event busy window of τ_i with switching time s
$I_L(s)$	Maximum interference from $hpL(i)$ with switching time s
$I_H(s, B_i^s(q,\delta_i))$	Maximum interference from $hpH(i)$ with switching time s
R_i	Worst-case response time of τ_i

function $\delta(q)$ is a pseudo super-additive[a] *function, which returns a lower bound on the time interval between the first and the last event of any sequence of $q+1$ event occurrences.*

The minimum distance function is an inverse description of upper arrival curve. For example, $\delta(k) = \Delta_k$ denotes that, the first and the last event of any sequence of $k+1$ events is at least Δ_k time units apart, i.e., $\overline{\alpha}(\delta(k)) = k+1$.

The concept of arrival curve or minimum distance function substationally generalizes conventional event stream models, such as sporadic, periodic, periodic with

[a]For pseudo super-additive we denote the property of a function δ that $\forall a, b \in \mathbb{N}^+ : \delta(a+b) \geq \delta(a) + \delta(b)$. It corresponds to the property of "good" arrival functions in [28].



(a) with varying j (b) with varying d

Fig. 1: The upper arrival curve of pjd event streams

jitter, and arbitrary event streams. For instance, for the arbitrary events modeled with the period p, the jitter j, and the minimum inter arrival distance d between successive two events, its upper arrival curve is

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

This arrival pattern is called pjd pattern that is often used as a type of the complex arrival pattern in many previous works [11, 29]. Some properties of pjd are shown in Fig. 1(a), from which we find that: when the jitter j increases, the initial burst increases; when d increases, the arrival interval between any two events increases; when $d = p$, the event arrival pattern is sporadic.

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

Definition 3. (Service Curve [11]) *Denote $C[s,t)$ as the available resource in the time interval $[s,t)$. Then, C, β^u and β^l represents the upper and lower bound on the resource available in any interval $t - s$, that is,*

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

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

As an arrival curve $\overline{\alpha}_i$ specifies the event and a service curve β specifies the available processing time, the event arrival curve $\overline{\alpha}_i(\Delta)$ has to be transformed to the *workload arrival curve* α_i to indicate the amount of computation time required for the arrived events in any time interval Δ. 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 $\overline{\alpha}_i^u = \alpha_i^u/c_i$, $\overline{\alpha}_i^l = \alpha_i^l/c_i$.

3.2. *System Settings*

Except the setting for task activations, our system settings are the same as a classic setting for the MCS in most previous papers [1,15,19]. Instead of the simple periodic or sporadic event stream, the task activations are modeled as an arbitrary event stream. The event arrival curve is used to model the upper bound of the arbitrary event stream.

In general, in our settings, a dual-criticality task set $\tau = \{\tau_1, ..., \tau_n\}$ is given to be scheduled on a uniprocessor. All tasks are independent. Each task, τ_i, is defined by its upper event arrival curve $\overline{\alpha}_i^u$, relative deadline D_i, WCET $\mathbf{C_i}$ and criticality L_i, where $\mathbf{C_i} = (C_i(LO), C_i(HI))$. We call the dual-criticality level as the LO criticality and the HI criticality. The WCETs of LO-critical tasks are modeled on the LO criticality, and the WCETs of HI-critical tasks are modeled on both criticalities. Namely, each LO-critical task only has a LO WCET, and each HI-critical task has a LO WCET and a HI WCET. For HI-critical tasks, the WCETs on the HI criticality are non-decreasing when compared to their WCETs on the LO criticality, i.e., $\forall \tau_i \in HI(\tau) : C_i(LO) \leq C_i(HI) \leq D_i$. This corresponds to the assumption that the execution time estimation on a higher criticality level is more conservative.

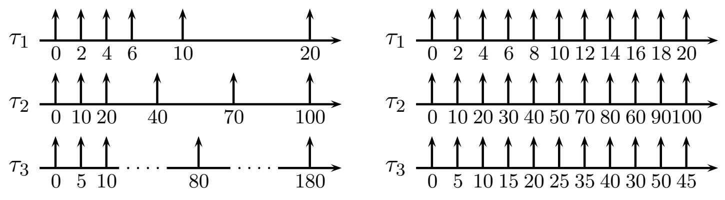
The system has two modes at runtime and is supposed to be scheduled as the adaptive run-time scheduling algorithm in [2]. That is, the system criticality starts in LO mode, where all tasks are assumed to not exceed/overrun their LO WCETs. If any task exceeds its LO WCET, then the system criticality transits immediately to the HI mode, where all LO-critical jobs or events are dropped and HI-critical tasks are assumed to be within their HI WCETs. The system is mixed-criticality schedulable if and only if all task deadlines can be met in LO mode and all HI-critical task deadlines can be met in HI mode. The system can be safely switched back to LO mode at any time that the system becomes idle [30, 31].

For the ease of expression in the sequel, we provide some short notations. $hp(i)$ denotes the subset of all tasks with priorities higher than that of the task τ_i. $hpH(i)$ denotes the subset of HI-critical tasks with priorities higher than that of the task τ_i. $hpL(i)$ denotes the subset of LO-critical tasks with priorities higher than that of the task τ_i.

3.3. *Motivation Example*

An arbitrary activation pattern that is modeled as the arrival curve can also be represented by the sporadic pattern. Since the sporadic pattern defines a minimum inter-activation interval, an arbitrary activation pattern can be represented by the sporadic pattern by defining a minimum inter-activation interval. However, this representation is pessimistic because it admits more events than the maximum number of events that will actually arrive.

Example 1. *In a uniprocessor system, there are three tasks, as shown in the following (task activations are set as pjd patterns, whose upper arrival curve* $\overline{\alpha}_i^u$ *is presented in Eq. 1):*



(a) arrival curve model (b) sporadic model

Fig. 2: The as-early-as-possible event trace of two different models

τ_i	L_i	$C_i(LO)$	$C_i(HI)$	D_i	$\overline{\alpha}_i^u\ (p, j, d)$
τ_1	LO	3	-	7	(10, 30, 2)
τ_2	HI	5	10	35	(30, 50, 10)
τ_3	HI	20	40	300	(100,220,5)

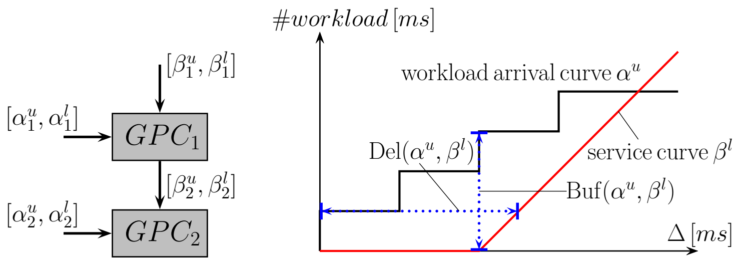
From $\overline{\alpha}_i^u$, the event trace that events arrive as early as possible is shown in Fig. 2(a). In order to apply the existing approaches to schedule this task set in MCSs*, those event traces should be modeled as the sporadic tasks. Since sporadic model only defines a period, the minimum distance between two activations of a task is used as the period. The as-early-as-possible event traces under the assumption of sporadic model is shown in Fig. 2(b). It is impossible to schedule this task set if events arrive as the assumption of sporadic model because the LO* WCET *of τ_1 is larger than the period of its sporadic model. However, in reality, the events will not arrive as frequently as the sporadic model assumes. By the sufficient busy-window schedulability test presented in Section 6.2, we find this task set is actually schedulable.*

4. Preliminaries

In this section, we introduce the well-established Modular Performance Analysis under the framework of Real-Time Calculus [12, 13] and the known Audsley's algorithm, which are the basis of the necessary and sufficient tests for verifying the schedulability of a given task set.

4.1. *Modular Performance Analysis*

In the framework of Real-Time Calculus, the task processing is often modeled by abstract performance component that acts as curve transformer in the domain of arrival and service curve, where the transferring function depends on the modeled processing semantics. The Greedy Processing Component (GPC) models a task that is triggered by the events which are queued up in the FIFO (first-in-first-out) buffer. For example, as shown in Fig. 3(a), there are two tasks that are abstracted as GPC_1 and GPC_2 in a preemptive fixed-priority scheduling system, where $[\alpha_i^u, \alpha_i^l]$



$[\beta_1^u, \beta_1^l]$
$[\alpha_1^u, \alpha_1^l]$
GPC_1
$[\beta_2^u, \beta_2^l]$
$[\alpha_2^u, \alpha_2^l]$
GPC_2
$\#workload\,[ms]$
workload arrival curve α^u
$\mathrm{Del}(\alpha^u, \beta^l)$
service curve β^l
$\mathrm{Buf}(\alpha^u, \beta^l)$
$\Delta\,[ms]$

(a) System abstraction of two tasks

(b) Computing $\mathrm{Del}(\alpha^u, \beta^l)$ and $\mathrm{Buf}(\alpha^u, \beta^l)$

Fig. 3: Modular performance analysis

and $[\beta_i^u, \beta_i^l]$ $(i = 1,\, 2)$ are the workload arrival curve and resource service curve of GPC_1 and GPC_2, respectively. Since the priority of GPC_1 is higher than GPC_2, GPC_1 will be always be served ahead of GPC_2. Thus, the lower service for GPC_2 is the system resource left over after serving the GPC_1. The maximum resource amount that needs to serve GPC_1 is represented by its upper arrival curve. Hence, the lower service for GPC_2 can be obtained by the following equation:

$$\beta_2^l(\Delta) \overset{\mathrm{def}}{=} \sup_{0 \le \lambda \le \Delta} \{\beta_1^l(\lambda) - \alpha_1^u(\lambda)\}, \tag{2}$$

where $\beta_1^l(\lambda)$ represents the system resource. In this paper, our platform is supposed to be with a dedicated unit-speed processor, then the system resource can be $\beta_1^u(\lambda) = \beta_1^l(\lambda) = \lambda$.

For any task, if its workload arrival curve and service curve are known, the response time of this task will be the largest interval gap between the workload arrival curve and service curve, and the maximum blocked workload will be the largest workload gap between them. For example, as shown in Fig. 3(b), with the provided lower service β^l and upper arrival workload α^u, the WCRT and the maximum workload of backlogged events for an event stream processed at a GPC can be computed as follows:

$$\mathrm{Del}(\alpha^u, \beta^l) \overset{\mathrm{def}}{=} \sup_{\lambda \ge 0} \big\{ \inf\{\tau \ge 0 : \alpha^u(\lambda) \le \beta^l(\lambda + \tau)\}\big\}, \tag{3a}$$

$$\mathrm{Buf}(\alpha^u, \beta^l) \overset{\mathrm{def}}{=} \sup_{\lambda \ge 0} \{\alpha^u(\lambda) - \beta^l(\lambda)\}, \tag{3b}$$

where Eq. 3a finds the largest horizontal gap between $\alpha^u(\Delta)$ and $\beta^l(\Delta)$ by searching the whole Δ and Eq. 3b finds the vertical gap between them by searching the whole workload. For more details about the two equations, we refer the readers to a formal introduction on the modular performance analysis in [11]. Note that, by applying Eqs. 3a, 3b, we get an upper bound of the WCRT, and the maximum backlogged workload. Then, to get the maximum number n_{max} of backlogged events, the WCET c of this task should be considered, i.e., $n_{max} = \lceil \mathrm{Buf}(\alpha^u, \beta^l)/c \rceil$.

Regarding to the system of two tasks as shown in Fig. 3(a), assume the deadlines for the two tasks are D_1 and D_2, this task set can be scheduled if and only if $\text{Del}(\alpha_1^u, \beta_1^l) \leq D_1$ and $\text{Del}(\alpha_2^u, \beta_2^l) \leq D_2$.

4.2. *Audsley's Algorithm*

Audsley's approach was proven by Dorin et al. in [19] as an optimal algorithm to assign the task priorities in MCSs. Audsley's algorithm starts with no task being assigned a priority. Priorities are assigned from the lowest to the highest, so that, at each step, a task that can be assigned with the lowest priority is selected out. Once a task is selected out, it is removed from the unassigned priority tasks, and Audsley's algorithm continues to assign the priority to the next task. Audsley's algorithm fails if there is no task that can be assigned with the lowest priority. The condition of using Audsley's algorithm to assign priorities [32] is that,

- The WCRT for a task τ_i can be determined by knowing which subset of tasks has higher priority than τ_i but without otherwise knowing what their specific priority assignments are.

Audsley's algorithm delivers an optimum priority assignment in a maximum of $n(n+1)/2$ steps. If Audsley's algorithm is not applicable, i.e., the above condition is not satisfied, the worst case for assigning priorities is to search over all $n!$ possible priority orderings.

5. A Necessary Test - NEC

This section presents the necessary test for verifying the schedulability of MCS scheduled by fixed-task-priority. The necessary test is based on the fact that a schedulable system should be able to schedule any event traces that comply with the arrival curves. Hence, we set up two necessary conditions that should hold for the system being schedulable. Audsley's algorithm is applied to search the priority assignment that makes the two conditions hold.

5.1. *Two Necessary Conditions*

Suppose that there are n tasks in an MCS, as shown in Fig. 4(a). We suppose two special situations, and the system should be schedulable under the following two situations.

- Condition LO: Suppose when the system is in LO criticality, event traces might occur in the worst-case patterns.
- Condition HI: Suppose there is no HI-critical event when the system is in LO mode; after the system enters into HI mode, HI-critical event traces might occur in the worst-case patterns.

When the MCS is in LO mode, no event will be dropped and the estimate of execution time will not change. The MCS in LO mode behaves as a non-MCS. Therefore, the Modular Performance Analysis in Section 4.1 can be used to verify the Condition LO. For the Condition HI, since no HI-critical event exists in LO mode, there will be no backlogged HI-critical events when the system enters into HI mode. In this situation, the MCS in HI mode can also be considered as a non-MCS on which only HI-critical tasks run. The Modular Performance Analysis is also used for verifying the condition HI. Note that the above two situations are two assumed situations that are just used for the necessary test.

5.2. *Test by Applying Audsley's Algorithm*

In many previous works [11, 13, 29], the Modular Performance Analyis is only used for analyzing the system with specific priority order. However, in order to apply Audsley's algorithm, the priority order of other tasks should have no effect on the WCRT of the lowest-priority task. Theorem 1 guarantees that the priority order has no effect on computing the WCRT of the lowest priority task, thus making Audsley's algorithm and the Modular Performance Analysis compatible for verifying the two necessary conditions.

In a system, suppose there are n tasks whose priorities need to be assigned so that all tasks can be schedulable, as shown in Fig. 4(a). For this system, we have the following theorem.

Theorem 1. *If the task τ_n is assigned the lowest priority without knowing the priority ordering of other tasks, the system can be abstracted as Fig. 4(b), where β_1^l is the lower service curve of the processor and other terms are the same as Section 3.2. When the system stays in LO mode, the lower service β_n^{lLO} for the task τ_n is bounded by*

$$\beta_n^{lLO}(\Delta) \overset{\text{def}}{=} \sup_{0\le\lambda\le\Delta} \Big\{ \beta_1^l(\lambda) - \sum_{j\in hp(n)} \overline{\alpha}_j^u(\lambda)\cdot C_j(LO) \Big\}. \tag{4}$$

When the system is in HI mode, with solely HI-critical tasks executing $C_i(HI)$ and no backlogged HI-critical events, the lower service β_n^{lHI} for the task τ_n is bounded by:

$$\beta_n^{lHI}(\Delta) \overset{\text{def}}{=} \sup_{0\le\lambda\le\Delta} \Big\{ \beta_1^l(\lambda) - \sum_{j\in hpH(n)} \overline{\alpha}_j^u(\lambda)\cdot C_j(HI) \Big\}. \tag{5}$$

Proof. Without the loss of generality, the priority for the task τ_i is ordered in a descending order, i.e., the priority of τ_i is greater than the priority of τ_j if $i < j$.

When the system is in LO mode, by iteratively using Eq. 2, we have

$$\beta_{i+1}^l(\Delta) = \sup_{0\le\lambda\le\Delta} \{\beta_i^l(\lambda) - \overline{\alpha}_i^u(\lambda)\cdot C_i(LO)\}, \forall\ i \le n-1,$$

where β_i^l is the lower service provided to the task τ_i. As

$$\beta_i^l(\Delta) = \sup_{0\le\lambda\le\Delta}\{\beta_{i-1}^l(\lambda) - \overline{\alpha}_{i-1}^u(\lambda)\cdot C_{i-1}(LO)\}, \forall\, i \le n-1,$$

we have

$$\begin{aligned}\beta_{i+1}^l(\Delta) &= \sup_{0\le\lambda\le\Delta}\{\beta_i^l(\lambda) - \overline{\alpha}_i^u(\lambda)\cdot C_i(LO)\}\\ &= \sup_{0\le\lambda\le\Delta}\Big\{\sup_{0\le\lambda'\le\lambda}\{\beta_{i-1}^l(\lambda') - \overline{\alpha}_{i-1}^u(\lambda')\cdot C_{i-1}(LO)\} - \overline{\alpha}_i^u(\lambda)\cdot C_i(LO)\Big\}.\end{aligned}$$

As $\overline{\alpha}_i^u(\lambda) \ge \overline{\alpha}_i^u(\lambda')$, we have

$$\begin{aligned}\beta_{i+1}^l(\Delta) &\le \sup_{0\le\lambda\le\Delta}\Big\{\sup_{0\le\lambda'\le\lambda}\{\beta_{i-1}^l(\lambda') - \overline{\alpha}_{i-1}^u(\lambda')\cdot C_{i-1}(LO)\} - \overline{\alpha}_i^u(\lambda')\cdot C_i(LO)\Big\}\\ &= \sup_{0\le\lambda\le\Delta}\Big\{\sup_{0\le\lambda'\le\lambda}\Big\{\beta_{i-1}^l(\lambda') - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda')\cdot C_j(LO)\Big\}\Big\}\\ &= \sup_{0\le\lambda\le\Delta}\Big\{\beta_{i-1}^l(\lambda) - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda)\cdot C_j(LO)\Big\}, \forall\, i \le n-1.\end{aligned}$$

Besides, as $\sup_{0\le\lambda'\le\lambda}\{f(\lambda')\} \ge f(\lambda)$, we have

$$\begin{aligned}\beta_{i+1}^l(\Delta) &\ge \sup_{0\le\lambda\le\Delta}\Big\{\beta_{i-1}^l(\lambda) - \overline{\alpha}_{i-1}^u(\lambda)\cdot C_{i-1}(LO) - \overline{\alpha}_i^u(\lambda)\cdot C_i(LO)\Big\}\\ &= \sup_{0\le\lambda\le\Delta}\Big\{\beta_{i-1}^l(\lambda) - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda)\cdot C_j(LO)\Big\}, \forall\, i \le n-1.\end{aligned}$$

Therefore, we have

$$\beta_{i+1}^l(\Delta) = \sup_{0\le\lambda\le\Delta}\Big\{\beta_{i-1}^l(\lambda) - \sum_{j=i-1}^{i}\overline{\alpha}_j^u(\lambda)\cdot C_j(LO)\Big\}. \tag{6}$$

Similarly, the index i in Eq. 6 can be extended from $n-1$ to 1, thus

$$\beta_n^{lLO}(\Delta) \overset{\text{def}}{=} \sup_{0\le\lambda\le\Delta}\Big\{\beta_1^l(\lambda) - \sum_{j\in hp(n)}\overline{\alpha}_j^u(\lambda)\cdot C_j(LO)\Big\}. \tag{7}$$

For the other priority settings, the lower service curve of Eq. 7 is unchanged, as long as the task τ_n is set with the lowest priority. Thus, the theorem holds.

Eq. 5 can also be proved in the same steps as proving Eq. 4. □

With β_n^{lLO} and β_n^{lHI}, Condition LO and Condition HI can be verified by checking

$$\text{Del}(\alpha_n^u(LO), \beta_n^{lLO}) \le D_n \tag{8a}$$

$$\text{Del}(\alpha_n^u(HI), \beta_n^{lHI}) \le D_n, \tag{8b}$$

where $\alpha_n^u(LO) = \overline{\alpha}_n^u \cdot C_n(LO)$ and $\alpha_n^u(HI) = \overline{\alpha}_n^u \cdot C_n(HI)$. If the task that is assigned the lowest priority is a LO-critical task, only Eq. 8a needs to be verified

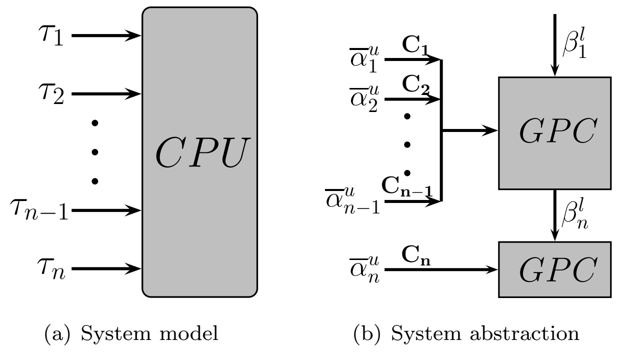


Fig. 4: A mixed-criticality system with n tasks

because the WCRT of this task does not need to be certified in HI mode. For the HI-critical task, Eqs. 8a, 8b need to be verified.

Audsley's algorithm searches the available task that can be assigned the lowest priority by checking Eqs. 8a, 8b. If Eqs. 8a, 8b hold, this task is selected out with the assigned priority and Audsley's algorithm continues to assign priorities to the left tasks. If not, Audsley's algorithm will check Eqs. 8a, 8b by assigning the lowest priority to another task. If no task can be assigned the lowest priority, the necessary test fails and this task set is not schedulable.

Example 2. *Returning to our motivation example, by using the necessary test, the task τ_1 can only be assigned with the highest priority. Its* WCRT *is 6. The task τ_2 can only be assigned with the second priority. Its* WCRT *is 20 in LO mode and 10 in HI mode. The task τ_3 is assigned with the lowest priority. Its* WCRT *is 139 in LO mode and 200 in HI mode. This task set passes the necessary test.*

6. Two Sufficient Tests

The necessary test can only guarantee that a task set is not schedulable if the necessary test fails, and cannot guarantee that a task set is schedulable if the necessary test does not fail. In this section, we present two sufficient tests towards the arbitrary activated tasks. The task set that succeeds with the sufficient test is schedulable.

The idea of sufficient tests is to verify whether the upper bound of a task response time is smaller than this task's relative deadline. If so, this task is deemed schedulable. The task is classified to be the LO-critical task or the HI-critical task. Since the LO-critical task only runs in LO mode and the MCS in LO mode can be considered as a non-MCS, the response time is bounded by $\mathrm{Del}(\alpha_n^u, \beta_n^{lLO})$ in Eq. 8a. Hence, the Eq. 8a is also the sufficient verification for LO-critical tasks. For the HI-critical task, however, computing the upper bound of response time is not so straightforward.

In the following, we present two approaches to compute the upper bound of response time of HI-critical tasks. The first approach is called the workload arrival curve approach. Suppose a task is set with the lowest priority. By deriving the workload arrival curve of all higher-priority tasks in both LO and HI modes, the lower bound of provided service to the lowest-priority task is derived. Then, the response time of the lowest-priority task can be bounded by applying the Modular Performance Analysis. The second approach is to apply the busy-window analysis to compute the upper bound of response time. In non-MCS systems, the busy-window analysis allows to calculate an upper bound on the time interval the processor is busy processing a task τ_i and its interferences $hp(i)$. We extend the busy-window analysis to MCSs to analyze the upper bound of response time of a HI-critical task.

6.1. *Workload Arrival Curve Analysis - WAC*

The idea of this approach is to derive a workload arrival curve $\alpha^u_{hp(i)}$ that upper bounds the workload of all tasks with higher priorities than the HI-critical task τ_i in both modes, including a mode switch. Hence, the remaining service for the task τ_i can be safely bounded by using Eq. 2, and the WCRT can be computed by using Eq. 3a.

The system starts in LO mode. Before the mode switch, the WCETs of all tasks are assumed to be $C_j(LO)$. Then, the workload arrival curve of tasks with higher priorities than τ_i is that

$$\alpha^{LO}_{hp(i)} \overset{\text{def}}{=} \sum_{j \in hp(i)} \overline{\alpha}^u_j \cdot C_j(LO).$$

Assume for the task τ_j, there are $\mathrm{Buf_j}$ events that are backlogged when the mode switch is triggered. Then, in HI mode, as the LO-critical tasks are not executed and the WCETs for HI-critical tasks are assumed to be $C_j(HI)$, we have

$$\alpha^{HI}_{hp(i)} \overset{\text{def}}{=} \sum_{j \in hpH(i)} (\overline{\alpha}^u_j + \mathrm{Buf}_j) \cdot C_j(HI). \tag{9}$$

To safely bound $\alpha^{HI}_{hp(i)}$, the maximum number of events $\mathrm{Buf}^{\max}_j$ that can be backlogged in LO mode is used. The computation of $\mathrm{Buf}^{\max}_j$ indicates the case that the task τ_j receives the interference from all the other tasks in $hp(i)$, i.e., the task τ_j is set with the lowest priority in $hp(i)$. For each task in $hp(i)$, by setting its priority as the lowest in $hp(i)$, $\mathrm{Buf}^{\max}_j$ can be computed by using Eqs. 3b, 4, i.e.,

$$\mathrm{Buf}^{\max}_j = \left\lceil \frac{\mathrm{Buf}\big(\alpha^u_j(LO), \beta^{lLO}_j\big)}{C_j(LO)} \right\rceil. \tag{10}$$

Since $\mathrm{Buf}^{\max}_j$ is rounded up, the released but not finished event at the mode switch is also included in $\mathrm{Buf}^{\max}_j$. As the backlogged events cannot be over $\mathrm{Buf}^{\max}_j$ and the released workload in HI mode cannot be over $\overline{\alpha}^u_j \cdot C_j(HI)$, $\alpha^{HI}_{hp(i)}$ in Eq. 9 is an upper bound of the workload in HI mode.

To get a workload arrival curve that upper bounds the workload of $hp(i)$ in both modes, the following theorem can be used.

Theorem 2. *For an* MCS *with a setting described in Section 3.2. The workload arrival curve that upper bounds the workload of $hp(i)$ in both modes can be computed by the following equation:*

$$\alpha^u_{hp(i)}(\Delta) \overset{\text{def}}{=} \sup_{0 \le \lambda \le \Delta} \Big\{ \sum_{j \in hp(i)} \overline{\alpha}^u_j(\Delta - \lambda) \cdot C_j(LO) + \sum_{j \in hpH(i)} (\overline{\alpha}^u_j(\lambda) + \text{Buf}^{\max}_j) \cdot C_j(HI) \Big\}. \tag{11}$$

Proof. We consider a time interval $[s, t)$ with $t - s = \Delta$, and set t_c as the time of mode switch. There are three possibilities, i.e.,

(1) $t \le t_c$, the system stays in LO mode.
(2) $s \ge t_c$, the system stays in HI mode.
(3) $s < t_c < t$, the system travels from LO mode to HI mode.

Since $\alpha^{LO}_{hp(i)} \le \alpha^u_{hp(i)}$ and $\alpha^{HI}_{hp(i)} \le \alpha^u_{hp(i)}$, Eq. 11 holds for the case 1 and the case 2.

For the case 3, we set $aw[s, t)$ the arrival workload in the interval of $[s, t)$. We have

$$\begin{aligned} aw[s,t) &= aw[s,t_c) + aw[t_c,t) \\ &\quad \text{Subst. } \lambda = t - t_c \\ &= aw[s, s+\Delta-\lambda) + aw[s+\Delta-\lambda, s+\Delta) \\ &\le \alpha^{LO}_{hp(i)}(\Delta - \lambda) + \alpha^{HI}_{hp(i)}(\lambda) \\ &\le \alpha^u_{hp(i)}(\Delta) \end{aligned}$$

Hence, the bound computed by Eq. 11 safely bounds the workload arrival curve in both modes. □

With the $\alpha^u_{hp(i)}$, the lower bound of remaining service for the HI-critical task τ_i can be computed with Eq. 2, i.e.,

$$\beta^l_i(\Delta) \overset{\text{def}}{=} \sup_{0 \le \lambda \le \Delta} \Big\{ \beta^l_1(\lambda) - \alpha^u_{hp(i)}(\lambda) \Big\}.$$

Then, to compute the WCRT of τ_i, every job of the HI-critical task τ_i is assumed to be executed with the $C_i(HI)$. This assumption sufficiently bounds the workload of this task. Thus, the workload arrival curve of τ_i is $\alpha^u_i = \overline{\alpha}^u_i \cdot C_i(HI)$. This HI-critical task τ_i can be scheduled if $\text{Del}(\alpha^u_i, \beta^l_i) \le D_i$.

Since there is no assumption in the priority ordering of the task set $hp(i)$ in this approach, the condition of applying Audsley's algorithm is satisfied. Audsley's algorithm is applicable for this approach.

Example 3. *Returning to the motivation example, by deriving the workload arrival curve of $hp(3)$ of applying Eq. 11, the* WCRT *of τ_3 is 338. By deriving the* *workload arrival curve of* $hp(1)$*, which is* $\overline{\alpha}_1 \cdot C_1(LO)$*, the* WCRT *of* τ_2 *is 37. This task set cannot be scheduled.*

6.2. *Busy-Window Analysis - BW*

In the non-MCS, the busy-window analysis allows to calculate an upper bound on the time interval that the processor is busy processing a task τ_i and its interferences from $hp(i)$ [33, 34]. Based on the busy-window, one can calculate an upper bound on a task's WCRT. In this section, we present a method that can sufficiently bound the response time of a HI-critical task in the MCS by applying the busy-window analysis. It first computes the maximum time to process any q HI-critical events of a task, based on which the upper bound of this task's WCRT is calculated.

Similar to the busy-window formulation of equation 3 in [34], the multi-event busy-window $B_i(q, \delta_i)$ [b] is defined.

Definition 4. (Multi-Event Busy-Window) *Assuming the processor is initially idle, the multi-event busy-window* $B_i(q, \delta_i)$ *describes an upper bound on the amount of time that a resource requires to serve* q *activations of the HI-critical task* τ_i *in* MCS*s.*

During processing q activations of the HI-critical task τ_i, there can be two cases:

- case 1: the MCS always stays in LO mode.
- case 2: the MCS transits from LO mode to HI mode, or completely stays in HI mode.

For the case 1, the multi-event busy-window is denoted as $B_i^{LO}(q, \delta_i)$, which can be obtained by calculating the following formula until convergence [34, 35].

$$B_i^{LO}(q, \delta_i) \overset{\text{def}}{=} q \cdot C_i(LO) + \sum_{j \in hp(i)} \overline{\alpha}_j(B_i^{LO}(q, \delta_i)) \cdot C_j(LO). \tag{12}$$

For the case 2, we define s as the time that the mode switch is triggered. s is restricted in the interval $[0,\, B_i^{LO}(q, \delta_i))$, because all q events would have been finished before the mode switch if $s \geq B_i^{LO}(q, \delta_i)$. If $s = 0$, it means that the MCS completely stays in HI mode. Denote $B_i^s(q, \delta_i)$ as the multi-event busy-window that the mode switch is triggered at s. Compared with $B_i^{LO}(q, \delta_i)$ in Eq. 12, the computation of $B_i^s(q, \delta_i)$ should separately consider LO-critical tasks and HI-critical tasks because LO-critical tasks can interfere τ_i only in LO mode. Hence, we formulate $B_i^s(q, \delta_i)$ as follows:

$$B_i^s(q, \delta_i) \overset{\text{def}}{=} q \cdot C_i(HI) + I_L(s) + I_H\big(s, B_i^s(q, \delta_i)\big), \tag{13}$$

where $I_L(s)$ refers to the maximum interference from $hpL(i)$ in the interval $[0, s)$, and $I_H(s, B_i^s(q, \delta_i))$ refers to the maximum interference from $hpH(i)$ in the interval $[0, B_i^s(q, \delta_i))$.

[b] δ_i is the minimum distance function of the HI-critical task τ_i

As LO-critical tasks are prevented from executing after s, the maximum interference from $hpL(i)$ is bounded by:

$$I_L(s) \overset{\text{def}}{=} \sum_{j \in hpL(i)} \overline{\alpha}_j(s) \cdot C_j(LO). \tag{14}$$

Regarding to the computation of $I_H(s, B_i^s(q, \delta_i))$, we first compute the maximum interference of every task in $hpH(i)$, and accumulate them together. Consider a specific task $\tau_k \in hpH(i)$ and use $I_k(s,t)$ to denote the maximum interference of this task. The maximum number of events within $[0,t)$ is $\overline{\alpha}_k(t)$. Suppose $C_k(HI)$ is the workload due to the release of m events in $[s,t)$ and $\overline{\alpha}_k(t) - m$ events are executed by $C_k(LO)$. Hence,

$$I_k(s,t) = m \cdot C_k(HI) + (\overline{\alpha}_k(t) - m) \cdot C_k(LO).$$

To maximize $I_k(s,t)$, m should be as large as possible because $C_k(HI) \geq C_k(LO)$. There are two constraints on m. First, m should be less than the maximum number of arrival events during $[0,t)$, i.e., $m \leq \overline{\alpha}_k(t)$. Second, m should be also less than the sum of backlogged events at time s and arrival events during $[s,t)$, i.e., $m \leq \overline{\alpha}_k(t-s) + \mathrm{Buf}_k(s)$, where $\mathrm{Buf}_k(s)$ is the maximum backlogged events at time s. We use $X_k(s,t)$ to denote the maximum m and $Y_k(s,t)$ to denote the number of events that are executed by $C_k(LO)$. Therefore,

$$X_k(s,t) \overset{\text{def}}{=} \min\left\{\mathrm{Buf}_k(s) + \overline{\alpha}_k(t-s),\ \overline{\alpha}_k(t)\right\}, \tag{15a}$$

$$Y_k(s,t) \overset{\text{def}}{=} \overline{\alpha}_k(t) - X_k(s,t). \tag{15b}$$

Exactly computing the $\mathrm{Buf}_k(s)$ is difficult as $\mathrm{Buf}_k(s)$ depends on the specific event arrivals of τ_k and $hp(k)$. Here, we provide an upper bound, i.e.,

$$\mathrm{Buf}_k(s) \overset{\text{def}}{=} \min\{\overline{\alpha}_k(s), \mathrm{Buf}_k^{max}\}, \tag{16}$$

where $\mathrm{Buf}_k^{\max}$ is computed with Eqs. 3b, 4 by setting τ_k with the lowest priority in $hp(i)$, which is the same as the Eq. 10 in deriving the workload arrival curve. Note that $X_k(s,t)$ is a valid upper bound on the events executed in $[s,t)$, but $Y_k(s,t)$ is neither an upper bound nor a lower bound. Nevertheless, the computed $I_k(s,t)$ with $X_k(s,t)$ and $Y_k(s,t)$ is an upper bound. Then, the maximum interference $I_H(s,t)$ is

$$I_H(s,t) \overset{\text{def}}{=} \sum_{k \in hpH(i)} \left\{X_k(s,t) \cdot C_k(HI) + Y_k(s,t) \cdot C_k(LO)\right\}. \tag{17}$$

With $I_L(s)$ and $I_H(s, B_i^s(q,\delta_i))$, $B_i^s(q,\delta_i)$ can be computed by iteration. Then, the $B_i(q,\delta_i)$ is the maximum $B_i^s(q,\delta_i)$ over all possible s, i.e.,

$$B_i(q,\delta_i) \overset{\text{def}}{=} \max(B_i^s(q,\delta_i)) \forall s,\ s \in [0, B_i^{LO}(q,\delta_i)).$$

To compute $B_i(q,\delta_i)$, s should be scanned. But it is not necessary to scan every s within $[0, B_i^{LO}(q,\delta_i))$. After an examination over $I_L(s)$ and $I_H(s,t)$, we find that only the points at which $\overline{\alpha}_j(s),\ \forall j \in hp(i)$ changes need to be checked.

Proposition 1. *$B_i^s(q, \delta_i)$ can only increase at the points where $\overline{\alpha}_j(s)$ changes $\forall j \in hp(i)$.*

Proof. Suppose s_1 and s_2 are two successive points that $\overline{\alpha}_j(s)$ changes $\forall j \in hp(i)$, and $s_1 < s_2$. Since $\overline{\alpha}_j(s)$ does not change within (s_1, s_2), $I_L(s)$ will not change within (s_1, s_2). When s increases from s_1 to s_2, from Eqs. 15a, 16, it can be known that $X_k(s,t)$ may decrease. Hence, $I_k(s,t)$ may also decrease, which leads to the decrease of $I_H(s,t)$. Therefore, $B_i^s(q, \delta_i)$ will also decrease when s increases from s_1 to s_2. We have $B_i^s(q, \delta_i) \leq B_i^{s_1}(q, \delta_i),\ \forall\ s \in (s_1, s_2)$. This means that $B_i^s(q, \delta_i)$ can only increase at the points where $\overline{\alpha}_j(s)$ changes $\forall j \in hp(i)$. □

Since within any two successive points, $B_i^s(q, \delta_i)$ will become smaller. To get $B_i(q, \delta_i)$, only the points at which $\overline{\alpha}_j(s),\ \forall j \in hp(i)$ changes need to be checked. [c]

With $B_i(q, \delta_i)$, we know the WCRT $R_i(q)$ of the q-th job is bounded by

$$R_i(q) \overset{\text{def}}{=} B_i(q, \delta_i) - \delta_i(q-1), \tag{18}$$

where $\delta_i(0)$ is set to be 0.

The computation of multi-event busy-window $B_i(q, \delta_i)$ assumes that all q events arrive earlier than the completion of their prior jobs (the $(q-1)$-event busy-time), i.e.,

$$\delta_i(q-1) \leq B_i(q-1, \delta_i).$$

We denote the maximum number of events that can be in a multi-event busy-window as Q_i, where Q_i is the last event that arrives earlier than the completion of its prior job, i.e.,

$$Q_i \overset{\text{def}}{=} \max\left(n : \forall q \in \mathbb{N}^+, q \leq n : \delta_i(q) \leq B_i(q, \delta_i)\right).$$

Then, the WCRT R_i of the task τ_i can be found among the Q_i events, i.e.,

$$R_i \overset{\text{def}}{=} \max_{q \in [1, Q_i]} (R_i(q)).$$

The task τ_i can be scheduled if and only if $R_i \leq D_i$. As R_i can be determined by knowing $hp(i)$ and without knowing their specific priority assignments, Audsley's algorithm can be used to check the schedulability of a task set by this approach.

Example 4. *We now use the task set in the motivation example as a running example to explain the procedures of using busy-window analysis. It can be observed that only τ_3 is possible to be assigned with the lowest priority. Suppose τ_3 is set with the lowest priority, the other higher priority and HI-critical task is τ_2. By Eq. 10, we get that the maximum number of backlogged events of τ_2 in LO mode is 2, i.e., $\mathrm{Buf}_2^{max} = 2$. The* WCRT *of τ_3 is computed by applying the following rounds.*

[c]If $hp(i)$ is empty, i.e., task τ_i is set as the highest priority, s should be 0. This is because the WCRT will be the largest if every event of this task is executed with a large WCET estimation.

In the first round, only one event of τ_3 is considered, i.e., $q = 1$. We first compute the busy-window $B_3(1, \delta_3)$ in LO mode. By Eq. 12, $B_3(1, \delta_3) = 78$. According to Proposition 1, only the points in $[0, 78)$ where $\overline{\alpha}_i^u$, $i \in 1, 2$ change need to be checked. We use s^* to denote those points. Then, for every s^*, $I_L(s^*)$ of Eq. 14 is computed. With $\mathrm{Buf}_2^{max} = 2$ *and by successively applying Eqs. 16, 15a, 15b, 17, 13, $B_3^{s^*}(1, \delta_3)$ can be computed. The maximum $B_3^{s^*}(1, \delta_3)$ among all s^* is picked out. We get that $B_3^s(1, \delta_3) = \max(B_3^{s^*}(1, \delta_3)) = 140$. Since $\delta_3(0) = 0$, the* WCRT *$R_3(1)$ of one event is 140.*

In the second round, two events of of τ_3 are considered, i.e., $q = 2$. By the same computing steps as the first round, we get that $B_3^s(2, \delta_3) = 207$. As $\delta_3(1) = 5$, the WCRT *$R_3(2)$ is that $R_3(2) = B_3^s(2, \delta_3) - \delta_3(1) = 202$.*

We continue to increase q by one in every round and compute $R_3(q)$ in every round, until we find that $\delta_i(q-1) > B_i(q-1, \delta_i)$. In the motivation example, we find that, when $q = 10$, $B_3^s(10, \delta_3) = 747$. The earliest arrival time of the 11-th event is 780, which is greater than 747. It indicates that, the workload of first 10 events has no effect on the WCRT *of 11-th event. The 11-th event can be reconsidered as $q = 1$. Therefore, the maximum $R_3(q)$ where $q \leq 10$ represents the* WCRT *of the task τ_3. In the motivation example, the* WCRT *of τ_3 is 261. Therefore, we testify that τ_3 is schedulable by setting τ_3 with the lowest priority.*

The task τ_3 with the lowest priority can now be removed from this taskset. We continue to assign priorities to τ_1 and τ_2. For the HI-critical task τ_2, we follow the same computation procedures as computing the WCRT *of τ_3 to obtain the* WCRT *of τ_2. For the LO-critical task, we only need to apply Eq. 8a to get its* WCRT *because LO-critical tasks are not processed in HI mode. Overall, we find that, τ_1 with the highest priority and τ_2 with the second priority are schedulable.*

6.3. *Comparing WAC and BW*

6.3.1. *Complexity*

The computational overhead related to our schedulability tests can be attributed to two parts, i.e., the expense for searching feasible priority assignment, and the expense for verifying the schedulability of the task being assigned the lowest priority. Since WAC and BW apply Audsley's algorithm to search the feasible priority assignment, the overhead of the first part is the same for both approaches.

The computational overhead mainly depends on the second part, i.e., verifying whether a task can be assigned the lowest priority. If this task is a LO-critical task, WAC and BW use the same method as verifying Condition LO of the necessary test. If this task is a HI-critical task, the complexities of WAC and BW are different. There are four steps of applying WAC, which are, computing the maximum backlogged events $\mathrm{Buf}_j^{\max}$ for every task in $hp(i)$, computing the workload arrival curve $\alpha_{hp(i)}^u$ that upper bounds the workload of $hp(i)$, computing the lower bound of provided service β_i^l, and computing the WCRT by $\mathrm{Del}(\alpha_i^u, \beta_i^l)$. For applying B-W, there are two steps, which are, computing $\mathrm{Buf}_j^{\max}$ of every task in $hp(i)$, and computing the response time of every event within a maximum busy-window. Both WAC and BW are the same in first step, but are different in other steps.

Among the different steps, the computational expense of WAC mainly depends on the operations of curves, for instance, the max-convolution used in deriving the $\alpha^u_{hp(i)}$. If the two curves are periodic, the computation expense depends on the least common multiple of the two periods. If they are aperiodic, the computational expense depends on the number of aperiodic segments. The computational expense of BW mainly depends on computing the WCRT of every event within a busy-window. Specifically, it depends on how large a busy-window is, how many events could be in the busy-window, and how many changes are there of $\overline{\alpha}_j(s)$, $\forall j \in hp(i)$.

In our simulations, the WAC, with the support of RTC/S tool [36], is often faster than the BW.

6.3.2. ***Tightness***

Comparing with WAC, BW sets more constraints in deriving the interference from $hp(i)$, which results in that BW is tighter than WAC on the schedulability test.

In BW, X_k of Eq. 15a and Y_k of Eq. 15b set a constraint that the maximum events within $[0, t)$ cannot exceed $\overline{\alpha}_k(t)$. Buf_k in Eq. 16 sets a constraint that the maximum backlogged events cannot exceed the arrival events before the mode switch and the worst-case backlogged events. Since s is constrained by Eq. 12, the interference from LO-critical tasks is also constrained. While for WAC, in order to integrate the framework of Real-Time Calculus to do the sufficient test, it does not explore so much constraints in deriving $\alpha^u_{hp(i)}(\Delta)$.

7. Schedulability Evaluation

In this section, towards the arbitrary event streams, we present the schedulability evaluation on our proposed analyzing approaches. In specific, the proposed approaches are:

- NEC: The necessary test in Section 5, showing the necessary conditions that tasks should meet in order to be scheduled by fixed priority. NEC provides an upper bound for the sufficient tests, because task sets that are tested schedulable by any sufficient test are deemed to pass this test.
- WAC: The sufficient test by deriving the workload arrival curve in Section 6.1.
- BW: The sufficient test by the busy-window analysis in Section 6.2.

Besides, for the sporadic tasks, an existing approach is used to compare with our proposed approaches.

- AMC-max: The sufficient test by the most powerful response-time calculation for fixed-priority scheduling from [2]. This method is only valid for sporadic tasks whose relative deadlines are smaller than periods.

The RTC/S tool is used to the NEC and WAC test. It is also used to compute the maximum number of backlogged events in Eq. 10 for BW test. All the results are obtained from a simulation host with Intel i7-4770 processor and 16GB RAM.

7.1. *Task Set Generation*

The task set is generated in the same way as [6]. A random task set is generated by starting with an empty task set $\tau = \emptyset$, where random tasks are successively added. Although the four approaches can handle the task with any activation pattern, for the easiness, the task is set as pjd pattern in our simulations. By artificially varying the parameters, the effects of jitter and burst on the system schedulability are evaluated.

The random task set is generated as follows:

- The task utilization is a value of $(x + 0.5)/30$, where $x \in \{0, 1, ..., 29\}$.
- The probability of a random task being a HI-critical task is $\mathcal{P}$.
- $C_i(LO)$ is drawn from the uniform distribution over $\{1, 2, ..., C_L^{\max}\}$. There are two settings for $C_L^{\max}$. The first setting is to set $C_L^{\max} = 10$, in order to make the generated tasks mostly be light tasks (low utilization). The second setting is to set $C_L^{\max} = 40$, in order to produce a task set mixed with light and heavy tasks.
- $C_i(HI)$ is drawn from the uniform distribution over $\{C_i(LO), C_i(LO)+1, ..., 4 \cdot C_i(LO)\}$ if $L_i = HI$.
- The period p_i is drawn from the uniform distribution over $\{C_i(L_i), C_i(L_i) + 1, ..., 200\}$.
- The jitter j_i is set as $\mathcal{X} \cdot p_i$, where $\mathcal{X} \in [0, 5)$.
- The minimum inter distance d_i is set as $\mathcal{Y} \cdot p_i$, where $\mathcal{Y} \in [0, 1)$
- The relative deadline is set as that $D_i(LO) = D_i(HI) = \mathcal{Z} \cdot p_i$, where $\mathcal{Z} \in [0, 5)$.

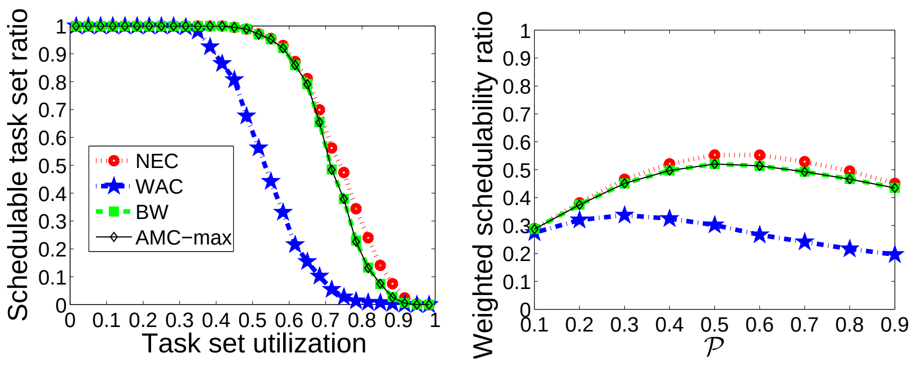
We introduce that

$$U_{LO}(\tau) \overset{\text{def}}{=} \sum_{\tau_i \in \tau} (C_i(LO)/p_i), \; U_{HI}(\tau) \overset{\text{def}}{=} \sum_{\tau_i \in \tau^H} (C_i(HI)/p_i), \tag{19}$$

where τ^H is the set of all HI-critical tasks in task set τ. The task set utilization is defined as $U(\tau) \overset{\text{def}}{=} (U_{HI} + U_{LO})/2$. For every task set generation, the utilization is allowed to be located in $[U^* - 0.005, U^* + 0.005]$, where U^* is a targeted utilization. If the generated utilization is smaller than $U^* - 0.005$, a new random task is added. If the generated utilization is greater than $U^* + 0.005$, this task set is discarded, and a new empty task set is started, until a task set with the allowed utilization is found.

7.2. *Evaluation Results*

In our experiments, for each target utilization, 1000 task sets were generated, and the schedulability of those task sets was determined by the four analyzing approach-



(a) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0, 0, 1)$ (b) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0, 0, 1)$

Fig. 5: Schedulability results towards the sporadic light task sets (all subfigures share the same color scheme)

es. The graphs are best viewed online in colour.

7.2.1. ***Schedulability Test on Sporadic Task Sets***

First, we evaluate the four schedulability test approaches towards the sporadic task sets with implicit deadlines.

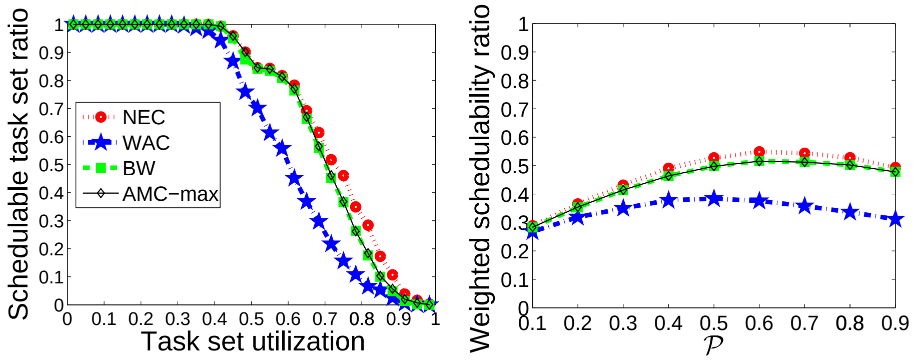
Fig. 5(a) and Fig. 6(a) show the percentage of light and mixed task sets that are tested schedulable. By setting $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0, 0, 1)$, tasks are generated as implicit-deadline sporadic tasks with 50% being HI-critial. In both figures, we observe that BW achieves exactly the same schedulable percentage as AMC-max, while outperforms the WAC by a large margin. This is expected as BW explores the same constraints as AMC-max to compute the WCRT. WAC pessimistically derives the workload arrival curve of higher-priority tasks, thus resulting in an overestimate of WCRT. Besides, we also observe that, the schedulable percentages of BW and AMC-max are slightly less than the limit illustrated by NEC upper bound, which indicates that schedulability tests of BW and AMC-max are very tight.

Next, we study the effects of the parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability by our proposed tests. To evaluate those parameters, the *weighted schedulability ratio* $W(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ [37] is used, which is defined as follows:

$$W(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) \overset{\text{def}}{=} \Big(\sum_{\forall\tau} U(\tau) \cdot S(\tau, \mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})\Big) / \sum_{\forall\tau} U(\tau),$$

where $S(\tau, \mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is the schedulability ratio of the task set whose utilization and parameters are $U(\tau)$ and $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$.

In order to generate implicit-deadline sporadic tasks, $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ must be $(0, 0, 1)$. Hence, only $\mathcal{P}$ can be investigated. The weighted schedulability ratio w.r.t., $\mathcal{P}$ is shown in Fig. 5(b) and Fig. 6(b). It shows that the achieved weighted



(a) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0, 0, 1)$ (b) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (0, 0, 1)$

Fig. 6: Schedulability results towards the sporadic mixed task sets (all subfigures share the same color scheme)

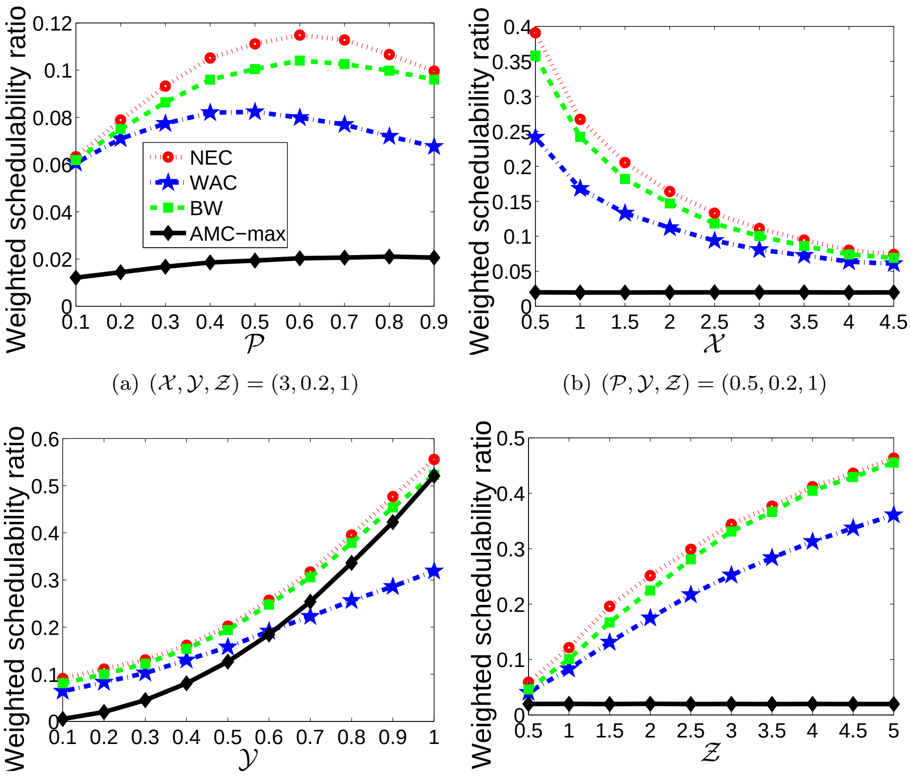
schedulability ratios of BW and AMC-max are exactly the same and are slightly lower than the upper bound of NEC. The performance gap between BW and NEC is small when $\mathcal{P}$ is small and becomes large when $\mathcal{P}$ increases. This is because, when $\mathcal{P}$ is small, most tasks are LO-critical. Since the schedulability test towards the LO-critical tasks are the same in the four approaches, i.e., Eq. 8a is used as the sufficient and necessary test, the schedulability ratio of the four approaches are expected to be close if $\mathcal{P}$ is small. From the four figures, it is observed that the superiority of BW/AMC-max over WAC is greater in scheduling light task sets than in scheduling mixed task sets.

7.2.2. ***Schedulability Test on Arbitrarily Activated Task Sets***

Except the schedulability evaluations on arbitrarily activated task sets, we also evaluate how the results are changed towards the arbitrarily activated task sets by varying the parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ (one parameter each time), whose meanings have been presented in Section 7.1.

Since the activation patterns of tested tasks are not sporadic and the relative deadlines are arbitrary, AMC-max cannot be directly used in testing the schedulability of those tasks. However, in a pessimistic way, sporadic model with implicit deadlines can still be used to model the tasks with arbitrary activations and arbitrary relative deadlines. Here is the way we take: suppose for a task τ_i, the minimum interval between any two task activations is d_i and its relative deadline is D_i. This task can be modeled as a sporadic task whose minimum inter distance and relative deadline is $\min(d_i, D_i)$. In this way, AMC-max can be used to test the schedulability of those tasks[d].

[d]If there are burst events for activating a task, i.e., $d_i = 0$, this task is unschedulabled by AMC-



(a) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (3, 0.2, 1)$

(b) $(\mathcal{P}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0.2, 1)$

(c) $(\mathcal{P}, \mathcal{X}, \mathcal{Z}) = (0.5, 3, 1)$

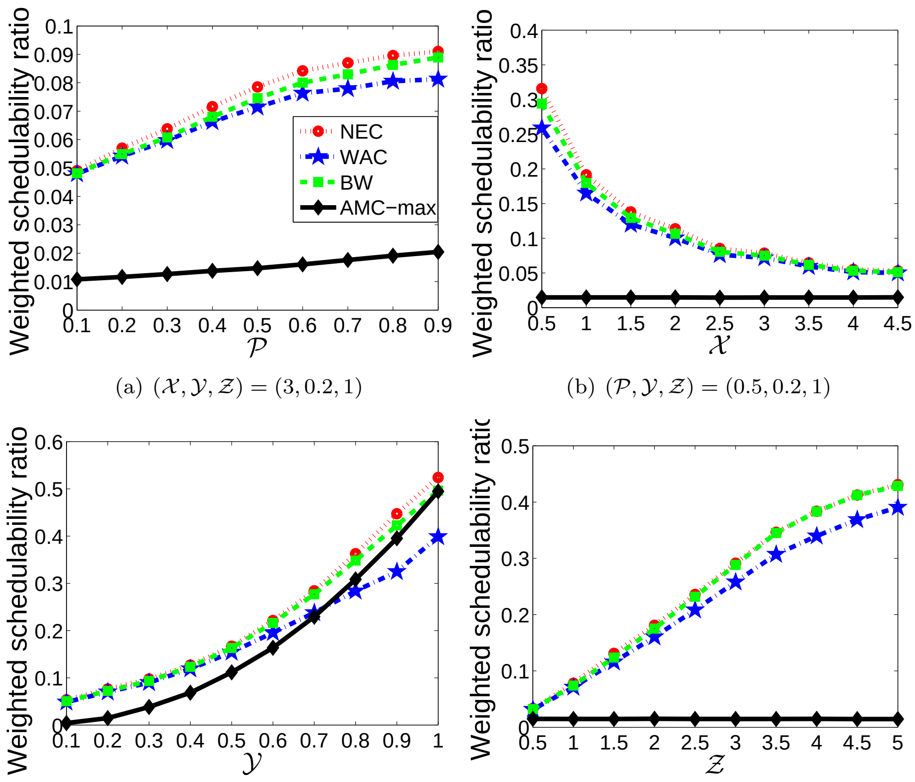
(d) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}) = (0.5, 3, 0.2)$

Fig. 7: The effects of parameters $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability towards light task sets (all subfigures share the same color scheme as the first figure)

Fig. 7 and Fig. 8 show the effects of $(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$ on the system schedulability towards light task set and mixed task set, respectively. Fig. 9 and Fig. 10 show the time expense of the proposed approaches to get one point of Fig. 7 and Fig. 8, respectively. From those figures, we can conclude that,

- Fig. 7(a) shows that, for the light task set, it is good to improve the system schedulability with the same portion of LO-critical and HI-critical tasks. Fig. 8(a) shows that, for the mixed task set, the system schedulability is better with a higher portion of HI-critical tasks. AMCmax performs the worst because it pessimistically estimate that there will be a task activation within any $0.2 \cdot p_i$. However, the truth is that the task can only be activated every $0.2 \cdot p_i$ in a short run and will be activated at least every p_i in the long run. In this way, AMCmax assumes more events than the reality in the long run. Our proposed

max test.



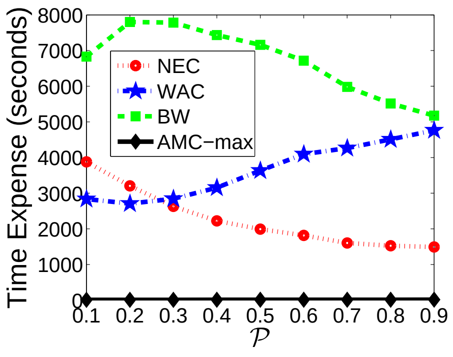
(a) $(\mathcal{X},\mathcal{Y},\mathcal{Z}) = (3, 0.2, 1)$ (b) $(\mathcal{P},\mathcal{Y},\mathcal{Z}) = (0.5, 0.2, 1)$

(c) $(\mathcal{P},\mathcal{X},\mathcal{Z}) = (0.5, 3, 1)$ (d) $(\mathcal{P},\mathcal{X},\mathcal{Y}) = (0.5, 3, 0.2)$

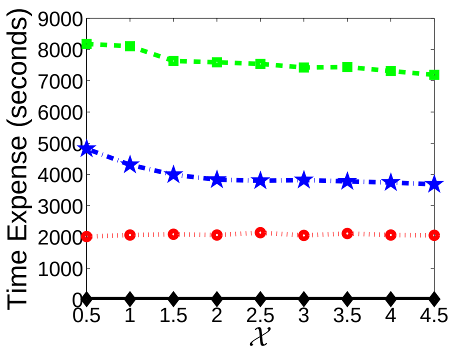
Fig. 8: The effects of parameters $(\mathcal{P},\mathcal{X},\mathcal{Y},\mathcal{Z})$ on the system schedulability (all subfigures share the same color scheme as the first figure)

approaches are able to incorporate this reality to the schedulability analysis, thus performing than AMCmax.

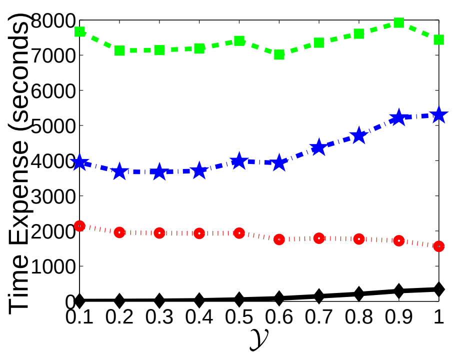
- Fig. 7(b) and Fig. 8(b) show that, the increase of jitter decreases the system schedulability. This is expected as the increase of jitter will also increase the number of task activations within a short interval. However, AMCmax does not change with increasing $\mathcal{X}$. This is because the jitter change will not affect the task minimum inter distance and relative deadline, thus it will not change the AMCmax estimation on task activations and deadlines.
- Fig. 7(c) and Fig. 8(c) show the increase of minimum inter-arrival distance between two events improves the system schedulability, because this parameter decreases the number of task activations within a certain interval. When $\mathcal{Y} = 1$, BW and AMC-max perform the same. This is expected because tasks become sporadic tasks with implicit deadlines when $\mathcal{Y} = 1$.
- Fig. 7(d) and Fig. 8(d) show that, the increase of relative deadline is helpful to



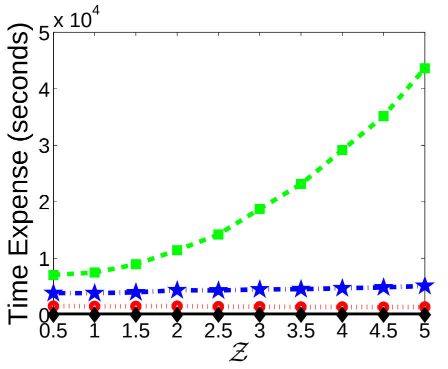
(a) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (3, 0.2, 1)$



(b) $(\mathcal{P}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0.2, 1)$



(c) $(\mathcal{P}, \mathcal{X}, \mathcal{Z}) = (0.5, 3, 1)$

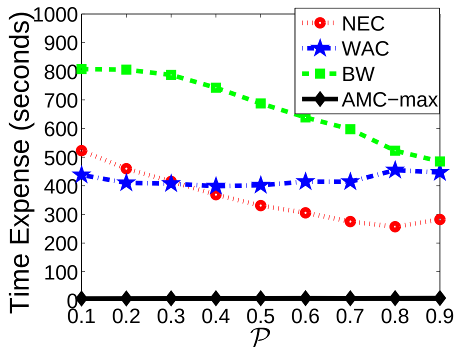


(d) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}) = (0.5, 3, 0.2)$

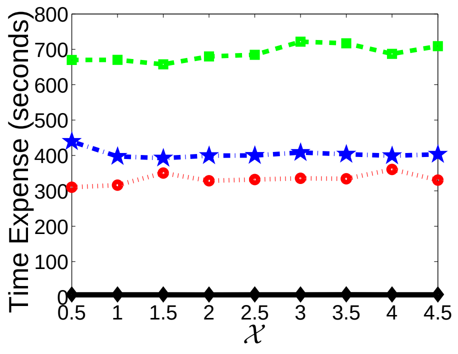
Fig. 9: The time expense of the schedulability test approaches towards light task sets (all subfigures share the same color scheme as the first figure)

increase the system schedulability. This is straightforward because increasing the relative deadline allows much longer response times, thus increasing the system schedulability. However, the increase of relative deadlines has no impact on the AMCmax. This is still because increasing relative deadlines does not change the pessimistic estimate of AMCmax towards those tasks.

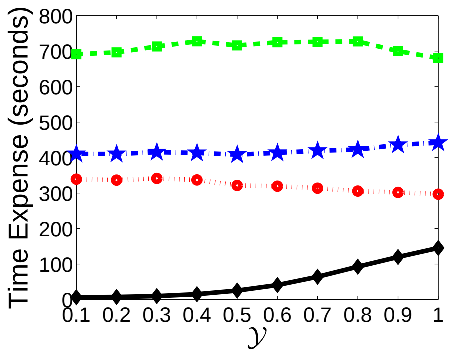
Overall, in all those figures, we observe that: the BW constantly outperforms the WAC, and keeps close results to NEC. This shows that BW is a very effective approach of analyzing the system schedulability towards the arbitrary activated tasks. Besides, from the performance of AMC-max, it can be observed that using sporadic task model to model arbitrary activated tasks leads to very pessimistic schedulability test results. Fig. 9 and Fig. 10 show that, AMC-max needs the least time to get the results and BW needs the most time to get the result.



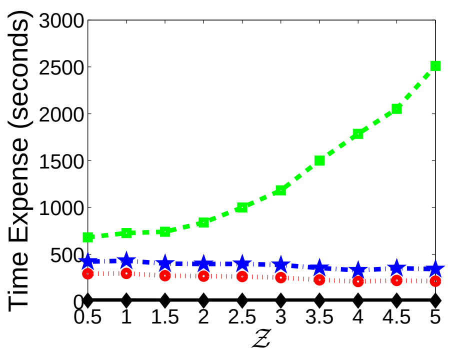
(a) $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = (3, 0.2, 1)$



(b) $(\mathcal{P}, \mathcal{Y}, \mathcal{Z}) = (0.5, 0.2, 1)$



(c) $(\mathcal{P}, \mathcal{X}, \mathcal{Z}) = (0.5, 3, 1)$



(d) $(\mathcal{P}, \mathcal{X}, \mathcal{Y}) = (0.5, 3, 0.2)$

Fig. 10: The time expense of the schedulability test approaches (all subfigures share the same color scheme as the first figure)

8. Conclusion

In this paper, we proposed to use the arrival curve to more accurately represent the upper bound of task activations; based on this representation, one necessary and two sufficient schedulability tests are proposed. The sufficient test of BW is proved to be more effective than the sufficient test of NEC. The experimental results show that, for scheduling implicit-deadline sporadic tasks, BW can achieve the same effectiveness as the state-of-the-art AMC-max. The results of BW are also close to the NEC result. which is an upper bound of schedulability result.

From the experimental results, we found that, the decrease of the jitter, the increase of minimum interval between any two task activations, and the increase of the relative deadlines will increase the system schedulability. Using sporadic tasks to model arbitrary tasks with arbitrary deadlines will result in very pessimistic schedulability test results, especially for scheduling tasks whose minimum task activation interval and relative deadlines are small.

References

1. Alan Burns and Rob Davis. Mixed criticality systems: A review. Technical report, 2013.
2. Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.
3. Sanjoy K Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Mixed-criticality scheduling of sporadic task systems. In *Algorithms–ESA 2011*, pages 555–566. Springer, 2011.
4. Pontus Ekberg and Wang Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 135–144, 2012.
5. Arvind Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Real-Time Systems Symposium (RTSS)*, pages 78–87, 2013.
6. Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-time systems*, 50(1):48–86, 2014.
7. Björn B Brandenburg, Hennadiy Leontyev, and James H Anderson. An overview of interrupt accounting techniques for multiprocessor real-time systems. *Journal of Systems Architecture*, 57(6):638–654, 2011.
8. Ken W Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.
9. ARINC Specification. 653-2: Avionics application software standard interface: Part 1-required services. Technical report, Avionics Electronic Engineering Committee, 2006.
10. Traub Matthias. *Durchgängige timing-bewertung von vernetzungsarchitekturen und gateway-systemen im kraftfahrzeug.* PhD thesis, Karlsruher Institut für Technologie (KIT), 2010.
11. Ernesto Wandeler. *Modular performance analysis and interface-based design for embedded real-time systems.* PhD thesis, ETH Zurich, 2006.
12. Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems (ISCAS)*, pages 101–104, 2000.
13. Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 190–195, 2003.
14. Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.
15. Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
16. Pengcheng Huang, Hoeseok Yang, and Lothar Thiele. On the scheduling of fault-tolerant mixed-criticality systems. In *ACM Design Automation Conference (DAC)*, pages 1–6, 2014.
17. Gang Chen, Kai Huang, Long Cheng, Biao Hu, and Alois Knoll. Dynamic partitioned cache memory for real-time mpsocs with mixed criticality. *Journal of Circuits, Systems and Computers*, 25(06):1650062, 2016.
18. Neil C Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.
19. François Dorin, Pascal Richard, Michaël Richard, and Joël Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time*

Systems, 46(3):305–331, 2010.
20. Sanjoy Baruah and Alan Burns. Implementing mixed criticality systems in ada. In *Reliable Software Technologies-Ada-Europe 2011*, pages 174–188. Springer, 2011.
21. Alan Burns and Sanjoy Baruah. *Timing faults and mixed criticality systems*, volume 6875. Springer Berlin Heidelberg, 2011.
22. Qingling Zhao, Zonghua Gu, and Haibo Zeng. Integration of resource synchronization and preemption-thresholds into edf-based mixed-criticality scheduling algorithm. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 227–236, 2013.
23. Qingling Zhao, Zonghua Gu, and Haibo Zeng. Pt-amc: Integrating preemption thresholds into mixed-criticality scheduling. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 141–146, 2013.
24. Fleming Thomas and Alan Burns. Extending mixed criticality scheduling. In *Proc. WMC, RTSS*, pages 7–12, 2013.
25. Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 145–154, 2012.
26. Moritz Neukirchner, Kai Lampka, Sophie Quinton, and Rolf Ernst. Multi-mode monitoring for mixed-criticality real-time systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 34:1–34:10, 2013.
27. Moritz Neukirchner, Philip Axer, Tobias Michaels, and Rolf Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Real-Time Systems Symposium (RTSS)*, pages 88–96, 2013.
28. Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet.* Springer, 2001.
29. Kai Lampka, Kai Huang, and Jian-Jia Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 267–276, 2011.
30. Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Real-Time Systems Symposium (RTSS)*, pages 13–23, 2011.
31. Haohan Li and Sanjoy Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Real-Time Systems Symposium (RTSS)*, pages 183–192, 2010.
32. Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.* Citeseer, 1991.
33. John P Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium (RTSS)*, pages 201–209, 1990.
34. Simon Schliecker, Jonas Rox, Matthias Ivers, and Rolf Ernst. Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 185–190, 2008.
35. Moritz Neukirchner, Sophie Quinton, Tobias Michaels, Philip Axer, and Rolf Ernst. Sensitivity analysis for arbitrary activation patterns in real-time systems. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 135–140, 2013.
36. Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.
37. Andrea Bastoni, Björn Brandenburg, and James Anderson. Cache-related preemption

and migration delays: Empirical approximation and impact on schedulability. *Proceedings of Operating Systems Platforms for Embedded Real-Time (OSPERT)*, pages 33–44, 2010.