Abstract—Mixed-criticality real-time scheduling has been developed to improve resource utilization while guaranteeing safe execution of critical applications. These studies use optimistic resource reservation for all the applications to improve utilization, but prioritize critical applications when the reservations become insufficient at runtime. Many of them however share an impractical assumption that all the critical applications will simultaneously demand additional resources. As a consequence, they under-utilize resources by penalizing all the low-criticality applications. In this paper we overcome this shortcoming using a novel mechanism that comprises a parameter to model the expected number of critical applications simultaneously demanding more resources, and an execution strategy based on the parameter to improve resource utilization. Since most mixed-criticality systems in practice are component-based, we design our mechanism such that the component boundaries provide the isolation necessary to support the execution of low-criticality applications, and at the same time protect the critical ones. We also develop schedulability tests for the proposed mechanism under both a flat as well as a hierarchical scheduling framework. Finally, through simulations, we compare the performance of the proposed approach with existing studies in terms of schedulability and the capability to support low-criticality applications.

I. INTRODUCTION

An increasing trend in embedded systems is towards open computing environments, where multiple functionalities are developed independently and integrated together on a single computing platform [1]. An important notion behind this trend is the safe isolation of separate functionalities, primarily to avoid fault containment. This raises the challenge of how to balance the conflicting requirements of isolation for safety assurance and efficient resource sharing for economical benefits. The concept of mixed-criticality appears to be important in meeting these two goals.

In many safety-critical systems, the correct behavior of some functionality (e.g., flight control) is more important (“critical”) to the overall safety of the system than that of another (e.g., in-flight cooling). In order to certify such systems as being correct, they are conventionally assessed under certain assumptions on the worst-case run-time behavior. For example, the estimation of Worst-Case Execution Times (WCETs) of code for highly critical functionalities involves very conservative assumptions that are unlikely to occur in practice. Such assumptions make sure that the resources reserved for critical functionalities are always sufficient. Thus, the system can be designed to be fully safe from a certification perspective, but the resources are in fact severely under-utilized in practice.

In order to close such a gap in resource utilization, Vestal [2] proposed the mixed-criticality task model that comprises of different WCET values. These different values are determined at different levels of confidence (“criticality”) based on the following principle. A reasonable low-confidence WCET estimate, even if it is based on measurements, may be sufficient for almost all possible execution scenarios in practice. In the highly unlikely event that this estimate is violated, as long as the scheduling mechanism can ensure deadline satisfaction for highly critical applications, the resulting system design may still be considered as safe.

To ensure deadline satisfaction of critical applications, mixed-criticality studies make pessimistic assumptions when a single high-criticality task executes beyond its expected (low-confidence) WCET. They assume that the system will either immediately ignore all the low-criticality tasks [3], [4], [5], [6], [7], [8] or degrade the service offered to them [9], [10], [11], [12]. They further assume that all the high-criticality tasks in the system can thereafter request for additional resources, up to their pessimistic (high-confidence) WCET estimates. Although these strategies ensure safe execution of critical applications, they have a serious drawback as pointed out in a recent article [9]. When a high-criticality task exceeds its expected WCET, the likelihood that all the other high-criticality tasks in the system will also require more resources is very low in practice. For instance, it is unlikely that adaptive cruise control and anti-lock braking, both of which are critical, would simultaneously require additional resources because their execution time depends on different inputs. Cruise control would most likely require additional resources when the cameras and lidars provide dense data, whereas the execution of anti-lock braking mainly depends on speed of the vehicle and friction on the tyres. Therefore, to penalize all the low-criticality tasks in the event that some high-criticality tasks require additional resources seems unreasonable.

In practice, most mixed-criticality systems are component-based wherein different vendors independently design and develop the various applications. For wide applicability, it is then natural that mixed-criticality scheduling strategies must consider the impact of WCET violations across component boundaries. To the extent possible, these strategies must limit this impact to within components, so that other components in the system can continue their execution uninterrupted. One extreme manifestation of this view is the reservation-based approach that completely isolates components but severely under-utilizes the resources. On the other hand, most of the recent mixed-criticality studies such as those mentioned above, completely ignore these component boundaries but still under-
utilize resources due to unrealistic assumptions.

**Contributions.** Addressing the two central issues described above, in this paper we propose a resource efficient mechanism to support low-criticality tasks while still ensuring isolation of high-criticality tasks. This mechanism comprises the following.

1) A new parameter to model the expected number of simultaneous violation of low-confidence WCET by high-criticality tasks.

2) A corresponding execution strategy that maximizes low-criticality task executions as long as this number is not exceeded.

3) To efficiently support component-based mixed-criticality systems, we employ our mechanism at the component level. We ensure that as long as the number of low-confidence WCET violations within a component does not exceed the component’s expected limit, task executions in other components, including low-criticality ones, remain unaffected.

It is worth noting that this mechanism generalizes both the reservation based approach in which high-criticality tasks are allocated resources based on their high-confidence WCETs [13], as well as the classical mixed-criticality studies that penalize all the low-criticality tasks (e.g., [8]). Considering a mixed-criticality scheduling strategy based on the Earliest Deadline First (EDF) policy (e.g., [6], [7], [8]), we also derive schedulability tests for the proposed mechanism. We derive these tests for a flat (non-hierarchical) as well as a hierarchical scheduling framework. While both these frameworks ensure isolation for high-criticality tasks as a result of employing criticality-aware scheduling, the hierarchical framework additionally supports compositionality, i.e., the ability of a system to derive properties (e.g., schedulability) for higher level components using derived properties of lower level components. We evaluate the performance of the proposed mechanism in terms of schedulability and the ability to support low-criticality executions through extensive simulations. These results show that our proposed mechanism outperforms all the other existing studies in terms of this dual objective.

**Related Work.** Since Vestal’s seminal work in 2007 [2], a growing number of studies have been introduced for mixed-criticality real-time scheduling, e.g., [3], [4], [5], [6], [7], [8], sharing the pessimistic strategy that all the low-criticality tasks will be immediately dropped upon WCET violation of a single high-criticality task. Some recent studies have presented solutions to improve support for low-criticality executions [9], [10], [11], [14], [12], [15]. The elastic mixed-criticality model allows for a flexible release pattern of low-criticality tasks depending on the runtime resource consumption of high-criticality tasks, essentially treating the low-criticality workload as background [11], [9], [10]. This was improved by the service adaptation strategy that decreased the dispatch frequency of low-criticality tasks only when a high-criticality task violated its low-confidence WCET. All the above studies however, share the unrealistic assumption that once a single high-criticality task violates its low-confidence WCET, all the other high-criticality tasks in the system will also exhibit similar behavior. The interference constraint graph strategy partially relaxes this assumption, at least in terms of its online strategy for penalizing low-criticality tasks [14]. The constraint graph is used to specify execution dependencies between high- and low-criticality tasks, and a response-time based approach was presented to determine graph constraints that improve low-criticality executions at runtime. However, it still uses high-confidence WCET estimates for all the high-criticality tasks when determining schedulability (test based on [2]), which again leads to the same unrealistic assumption and therefore results in resource under-utilization. Further, none of the above studies consider the impact of WCET violations in the context of component-based systems. A couple of recent studies proposed techniques to support hierarchical scheduling for component-based mixed-criticality systems [16], [13]. These studies focused on implementation issues however, and therefore did not consider the problems discussed above.

**II. System Model**

**A. Task and Component**

In this paper we consider constrained deadline mixed-criticality sporadic tasks (or tasks for short). Such a task can be specified as \( \tau_i = (T_i, L_i, C_i, D_i) \), where \( T_i \) denotes the minimum separation between job releases, \( L_i \) denotes the criticality level, \( C_i \) is a list of WCET values, and \( D_i \) (\( \leq T_i \)) denotes the relative deadline. We assume that tasks have only two criticality levels, \( LC \) denoting low-criticality and \( HC \) denoting high-criticality. Hence \( L_i \in \{ LC, HC \} \) and \( C_i = \{ C_i^L, C_i^H \} \), where \( C_i^L \) denotes \( LC \) WCET and \( C_i^H \) denotes \( HC \) WCET. If \( L_i = HC \), then \( \tau_i \) is called a \( HC \) task, otherwise \( \tau_i \) is called a \( LC \) task. We also assume that \( C_i^L < C_i^H \) for all the \( HC \) tasks, and \( C_i^L = C_i^H \) for all the \( LC \) tasks. Jobs of \( \tau_i \) are released with a minimum separation of \( T_i \) time units, and each job can execute for no more than \( C_i^H \) time units (\( C_i^L \) in the case of \( LC \) task) within \( D_i \) time units from its release. Let \( T = \{ \tau_1, \ldots, \tau_n \} \) denote a set of such mixed-criticality tasks that are scheduled on a single-core processor.

We assume that the tasks are partitioned into components, where each component \( C = (W, TL) \) comprises the following.

- A real time workload \( W \) denoting a subset of tasks from \( T \), and
- A HC Tolerance Limit \( TL \in \mathbb{N} \) denoting the maximum \( HC \) workload isolation limit of the component. As long as no more than \( TL \) tasks in the component simultaneously exhibit \( HC \) behavior (execution requirement is more than \( LC \) WCET), we must ensure that all the job deadlines in the other components, including those of \( LC \) jobs, are met. More details about this parameter are presented later in this section.

Partitioning the task set into components is mainly driven by practical considerations as mentioned in the introduction. Since these components are developed independently, it is desirable to limit the impact of WCET violations to within components as much as possible, while still efficiently utilizing the resources. The \( HC \) tolerance limit \( TL \) precisely does that in our model. It could be set based on component properties if information about the runtime behavior of \( HC \) jobs is available, e.g., probability of execution requirement exceeding \( LC \) WCET. It can also be determined such that the limit is maximized so as to support more \( LC \) job executions, while still...
maintaining system schedulability. \( C = (W, TL) \) is called a \( LC \) component if every task in its workload is a \( LC \) task, and for such components we assume that \( TL = 0 \). Otherwise, \( C \) is called a \( HC \) component.

**B. Task and Component Execution Model**

The execution semantics of a mixed-criticality task has been presented previously [3], and we summarize it as follows. A task \( \tau_i \) is said to be in low-criticality mode (or \( LC \) mode for short) as long as no job of the task has executed beyond its \( LC \) WCET \( C_i^L \). If \( \tau_i \) is a \( LC \) task, then this is the only available criticality mode. Whereas if \( \tau_i \) is a \( HC \) task, then it switches to high-criticality mode (or \( HC \) mode for short) at the time instant when any task requests to execute for more than its \( LC \) WCET. In \( HC \) mode, jobs of \( \tau_i \) can request to execute for no more than \( C_i^H \) time units.

We now define the execution semantics of a component \( C = (W, TL) \). \( C \) has two execution modes, an internal mode that concerns the behavior of tasks in \( C \), and an external mode that concerns the behavior of tasks in the other components. We first describe these two modes, and then discuss their implications.

**Internal Mode.** Component \( C \) experiences Internal Mode Switch (or IMS for short) at the earliest time instant when any \( HC \) task in \( C \) switches to \( HC \) mode. The component switches its internal mode from \( LC \) to \( HC \) at this time instant. Prior to this mode switch, all the task deadlines are required to be met. After this switch however, all the \( LC \) tasks in \( C \) can be dropped, and only the \( HC \) task deadlines are required to be met. There is no impact of this mode switch on the other components in the system.

**External Mode.** Component \( C \) experiences External Mode Switch (or EMS for short) at the earliest time instant when the \((TL + 1)\text{st} \) \( HC \) task in \( C \) switches to \( HC \) mode. The component switches its external mode from \( LC \) to \( HC \) at this time instant. Prior to this mode switch, at most \( TL \) tasks in \( C \) were executing in \( HC \) mode. After this switch however, all the \( HC \) tasks in \( C \) may execute in \( HC \) mode. Further, all the \( LC \) tasks in the system, including the \( LC \) tasks in the other components, are no longer required to meet deadlines. Component \( C \)'s internal as well as external modes could switch back to \( LC \) mode when there are no pending jobs in the system at some time instant.

Note that the intra- and inter-component execution requirements based on their internal and external modes respectively, are consistent with the mixed-criticality requirements in the existing literature (e.g., [6]). If \( C \) is a \( LC \) component, then its internal and external modes are identical and equal to \( LC \). On the other hand if \( C \) is a \( HC \) component, then these modes, together with the \( HC \) tolerance limit \( TL \), are key mechanisms for supporting \( LC \) job executions. If \( TL > 0 \), it is possible for IMS and EMS to occur at different time instants (asynchronously). Then, during the interval when component \( C \)'s internal mode is \( HC \) while its external mode is \( LC \), \( LC \) tasks in the other components are isolated from the internal mode switch of \( C \). That is, these \( LC \) tasks can continue their execution even though some \( HC \) tasks in \( C \) are already executing in \( HC \) mode.

The proposed model and execution strategy generalizes both the worst-case reservation based approach in which \( HC \) tasks are allocated resources based on their \( HC \) WCETs \([13]\), as well as the classical mixed-criticality studies that drop all the low-criticality tasks upon WCET violation by a single \( HC \) task (e.g., [8]). The former can be modeled by setting \( TL = |H| \), where \(|H| \) denotes the total number of \( HC \) tasks in the component, while the latter can be modeled by setting \( TL = 0 \).

**Scheduling Strategy.** In this paper we focus on the Earliest Deadline First (EDF) strategy, and assume that \( LC \) tasks are dropped (not considered for scheduling) once it becomes known that their deadlines are not required to be met. We have chosen this scheduling strategy because it has been successfully employed in the past for mixed criticality systems (e.g., [6], [7], [8]). To accommodate the sudden increase in demand when tasks start executing in \( HC \) mode, these existing studies artificially tighten the deadlines of \( HC \) tasks when they are executing in \( LC \) mode. This ensures that when a task switches to \( HC \) mode, it has some amount of time left until its real deadline to execute any additional demand. In this paper we assume that such deadline tightening strategies are employed.

For a task \( \tau_i = (T_i, L_i, C_i, D_i) \), we let \( D_i^L \) denote the artificially tightened deadline in \( LC \) mode of execution. By definition, \( D_i^L \leq D_i \) for all tasks, and \( D_i^L = D_i \) if \( L_i = LC \) because no tightening is required for such \( LC \) tasks. While a \( HC \) task \( \tau_i = (T_i, HC, C_i, D_i) \) is executing in \( LC \) mode, \( \tau_i \) must receive at least \( C_i^H \) processor units before its tightened deadline \( D_i^L \). When the task \( \tau_i \) switches to \( HC \) mode, it must receive at least \( C_i^H \) processor units before the actual deadline \( D_i \). Note that a \( HC \) task in component \( C \) that executes in \( LC \) mode after IMS of \( C \) will continue to be scheduled using its tightened deadline \( D_i^L \), unless it switches to \( HC \) mode. After EMS of \( C \) however, all the \( HC \) tasks are assumed to switch to \( HC \) mode and will be scheduled using their actual deadlines.

We consider two different scheduling frameworks in this paper; a flat (non-hierarchical) framework in which all the tasks in all the components are collectively scheduled by a single scheduler, and a hierarchical framework in which the tasks in components are scheduled by intra-component schedulers and the components themselves are scheduled by a inter-component scheduler. The flat framework is relevant in applications that do not use hierarchical scheduling (e.g., Deos Real-Time Operating System for avionics [17]), whereas the hierarchical framework is relevant in applications that require compositionality (e.g., ARINC653 in avionics [18]). Note that a criticality-aware flat scheduler also ensures isolation for high-criticality tasks, and hence from that perspective provides similar functionality as a hierarchical scheduler. In Section [III] we present the schedulability test under a flat scheduling framework, and in Section [IV] we present the schedulability test under a hierarchical scheduling framework. Finally, the capability of the proposed mechanism and the corresponding schedulability tests to support \( LC \) job executions are evaluated through extensive simulations in Section [V].
EDF-based flat scheduling framework. In Section III-A we denote the time instant for External Mode Switch or EMS of task \( J_i \) as \( r(J_i) \). If \( J_i \) itself. If \( J_i \) does not generate any demand before \( t_i \), it does not generate any demand after \( t_i \). Therefore \( \text{dbf}(\tau_i, t, t_i)_{[a]} \), defined as follows.

a: \( L_i = LC \),
b: \( L_i = HC \) and \( t - t_i < D_i - D_i^L \),
c: \( L_i = HC \) and \( t - t_i < D_i \), and

d: \( L_i = HC \) and \( D_i - D_i^L \) is a special case is when \( t_i \geq r(J_i^A) + D_i^L \) and \( r(J_i^A) + D_i > t_i \) (Figure 1(c)). In this case, \( J_i^A \) will not generate any demand after \( t_i \) according to Lemma 1. Thus, the demand of job \( J_i^A \) for the interval \([0, t)\) can be bounded as follows.

\[
\text{dbf}(J_i^A, t, t_i) = \begin{cases} 
C_i^L, & r(J_i^A) + D_i^L < t_i \\
C_i^H, & r(J_i^A) + D_i^L \geq t_i \\
\min\{t_i - r(J_i^A) \}, & r(J_i^A) + D_i < t_i \\
0, & r(J_i^A) + D_i^L > t_i 
\end{cases}
\]
If $\tau_i$ satisfies condition b, c or d, then $\tau_i$ is a HC task. Therefore, the total demand it generates is the sum of demand of all the jobs released before $t$. Among these jobs, the ones released before $r(J_i^A)$ will generate a demand of $C_i^L$, and the ones released after $r(J_i^A) + T_i$ will generate a demand of $C_i^H$. The demand of job $J_i^B$ itself is given in Equation 1 in the following lemmas we derive the $dbf$ of $\tau_i$ for the three conditions.

**Lemma 2:** If $\tau_i$ satisfies condition b ($t - t_i < D_i - D_i^L$), no job of $\tau_i$ can execute for $C_i^H$ time units. Therefore $\tau_i$ can generate maximal demand during $[0, t)$ if the first job of $\tau_i$ is released at time instant 0 and all the successive jobs are released as soon as possible.

**Proof:** We prove this lemma by contradiction. Suppose there exists a job $J_i$ of $\tau_i$ that can generate a demand of $C_i^H$ time units in the interval $[0, t)$. Then it must be true that $r(J_i^A) + D_i \leq t$ and $r(J_i) + D_i^L \geq t_i \Rightarrow t - t_i \geq D_i - D_i^L$, because $\tau_i$ is a HC task that switched to HC mode at $t_i$. This contradicts our assumption that $t - t_i < D_i - D_i^L$. Thus no job of $\tau_i$ that satisfies condition b can generate a demand of $C_i^H$ time units in the interval $[0, t)$. Therefore $\tau_i$ essentially behaves like a LC task, and this proves the lemma.

Thus $dbf(\tau_i, t, t_i)[b]$ is given as follows.

$$dbf(\tau_i, t, t_i)[b] = \left\lfloor \frac{t_i}{T_i} \right\rfloor C_i^L + dbf(J_i^A, t, t_i)$$

**Lemma 3:** If $\tau_i$ satisfies condition c ($t - t_i \geq D_i$), it generates maximal demand during $[0, t)$ if the first job of $\tau_i$ is released at $t - D_i - \lfloor (t - D_i)/T_i \rfloor \times T_i$, and all the successive jobs are released as soon as possible (scenario shown in Figure 2).

**Proof:** If $t - t_i \geq D_i$ and the first job of $\tau_i$ is released at $t - D_i - \lfloor (t - D_i)/T_i \rfloor \times T_i$, then the last job released before $t$ will have its deadline at $t$. In this case, $t_i$ happens before the release time of this last job. Therefore the last job can generate a demand of $C_i^H$ in the interval. Additionally, the number of jobs with deadline before $t$ as well as the number of jobs that can generate $C_i^H$ demand during $[0, t)$ are maximized with this pattern. This proves the lemma. ■

Intuitively speaking, the demand is maximized when the deadline of a job of $\tau_i$ coincides with $t$, because it maximizes the possible executions for $\tau_i$ in HC mode. Thus, $dbf(\tau_i, t, t_i)[c]$ is given as follows.

$$dbf(\tau_i, t, t_i)[c] = b_i C_i^L + dbf(J_i^A, t, t_i) + a_i C_i^H$$

where

$$b_i = \left\lfloor \frac{t_i - (t - D_i - \lfloor (t - D_i)/T_i \rfloor \times T_i)}{T_i} \right\rfloor$$

and

$$a_i = \left\lfloor \frac{t - D_i}{T_i} \right\rfloor - b_i$$

(5)

If $\tau_i$ satisfies condition d, it does not have a single execution pattern that maximizes its demand as stated in the following lemma.

**Lemma 4:** If $\tau_i$ satisfies condition d ($D_i - D_i^L \leq t - t_i < D_i$), it generates maximal demand if its first job is either released at 0 (condition b) or at $t - D_i - \lfloor (t - D_i)/T_i \rfloor \times T_i$ (condition c).

**Proof:** Since $D_i - D_i^L \leq t - t_i < D_i$, $\tau_i$ can have at most one job that can generate a demand of $C_i^H$ in the interval $[0, t)$. If the first job of $\tau_i$ is released at $t - D_i - \lfloor (t - D_i)/T_i \rfloor \times T_i$, and all the successive jobs are released as soon as possible (release pattern of condition c), then the last job is a special job $J_i^A$ and is the only job generating $C_i^H$ demand. The only way to further increase the demand of $\tau_i$ is to add a new job in the interval by shifting the pattern left to the point when the first job is released at time instant 0.

Thus, $dbf(\tau_i, t, t_i)[d]$ is given as follows.

$$dbf(\tau_i, t, t_i)[d] = \max \{ dbf(\tau_i, t, t_i)[b], dbf(\tau_i, t, t_i)[c] \}$$

(6)

C. $dbf$ of component $C$

In this section we present the $dbf$ of a component $C = \{W, TL\}$. Let $dbf(C, t, t_E, t_I)$ denote the $dbf$ of component $C$ for a given time interval length $t$, with mode-switch instants $t_I$ (IMS) and $t_E$ (EMS).

We first present $dbf$ for the case when $TL = 0$ and then for the case when $TL > 0$. Note that among all the HC tasks in $C$, at most $TL$ of them can switch to HC mode in the interval $[t_I, t_E)$, while all the remaining HC tasks are assumed to switch to HC mode at $t_E$.

If $TL = 0$, then this means $t_E$ (EMS) is equal to $t_I$ (IMS), because $C$’s internal and external modes will switch at the same time. Thus, each HC task $\tau_i$ in $C$ will switch to HC mode at $t_i = t_I = t_E$, and hence $dbf(C, t, t_E = t_I, t_I)$ is given as follows.

$$dbf(C, t, t_E = t_I, t_I) = \sum_{\tau_i \in C} dbf(\tau_i, t, t_I) \quad (TL = 0)$$

(7)

If $TL > 0$, then at most $TL$ HC tasks can switch to HC mode before $t_E$. To compute the $dbf$ of $C$, we then need to determine which HC tasks should switch to HC mode before $t_E$ so as to maximize the total demand. The following lemma asserts that for any HC task, its demand is maximized when it switches to HC mode either at $t_I$ or $t_E$.

**Lemma 5:** If a HC task $\tau_i$ switches to HC mode at some time $t_i \in [t_I, t_E]$, then $dbf(\tau_i, t, t_I)$ is maximized when $t_i$ is either equal to $t_E$ or $t_I$.
A tighter bound for the dbf of component \( C \) can be obtained using an optimization presented in Section 3 of the Appendix.

### D. Schedulability Test and Tolerance Limit

In this section we derive the schedulability test for a mixed-criticality system comprising multiple components and scheduled under a flat scheduling framework. Consider a system with \( p \) HC components \( C_1, C_2, \ldots, C_p \) and \( q \) LC components \( C_{p+1}, C_{p+2}, \ldots, C_{p+q} \). Each HC component \( C_i \) can independently switch its internal mode to HC at \( t_{1i} \). Once the first HC component switches its internal mode to HC at \( t_F \), all the LC tasks in the system are immediately dropped. We assume that all the HC tasks in the system can thereafter execute in HC mode.

Suppose there is a first deadline miss in the system at some time instant \( t \). Then, the total maximum demand generated by the system in \([0, t]\) must be greater than \( t \). This assertion immediately leads to the following theorem that presents the schedulability test.

**Theorem 1:** A mixed-criticality system comprising \( p \) HC components and \( q \) LC components is schedulable under a flat scheduling framework if,

\[
\sum_{i=1}^{p+q} \text{dbf}(C_i, t, t_F, t_{1i}) \leq t, \quad (9)
\]

where \( t_{MAX} \) is a pseudo-polynomial in the size of the input, and is defined in Section B of the Appendix.

The complexity of the schedulability test in Theorem 1 is exponential in the number of HC components, because we need to consider a separate internal mode switch instant for each component. In practice however, we expect the number of HC components scheduled on a single processor to be relatively small, and then the complexity of the proposed test is pseudo-polynomial in the size of the input. Besides, if there is freedom to select the allocation of system tasks to components, then it is feasible to create a component structure comprising only two components, while still fully supporting LC task executions. All the HC tasks in the system are allocated to a single HC component \( C_H = \{W_H, TL_H\} \), and each LC task can be either allocated to \( C_H \) or to a LC component \( C_L = \{W_L, TL_L = 0\} \). This two-component system is sufficient to consider all the possible design choices for isolating HC and LC task executions. This can be done by considering different values for the tolerance limit \( TL_H \), and by considering different allocations of LC tasks to the two components. We can choose the maximum possible value for these tolerance limit as long as the resulting system is still schedulable. Higher tolerance limit indicates support for more LC task executions, and thus better resource utilization. In Section 5 we show through simulations that our mechanism outperforms existing studies even with this two-component structure. However, if the allocation of tasks to components is fixed and the number of HC components is not small, then the hierarchical scheduling framework presented in the following section can be used to reduce the complexity of the test.

### IV. Schedulability Test for Hierarchical Scheduling Framework

Hierarchical scheduling has emerged as an effective mechanism to support temporal partitioning between applications, serving as a common scheduling paradigm in many mixed-criticality systems in practice [13]. It is preferred in practice because it supports compositionality so that higher-level properties can be derived from verified component-level properties. Therefore, to increase the practical relevance of the proposed mechanism, we develop a schedulability test under a hierarchical scheduling framework in this section.

#### A. Execution Strategy under Hierarchical Scheduling

For hierarchical systems, each component will have an additional parameter \( S \) denoting its local scheduler. We specify such a component as \( C = (\mathcal{V}, TL, S) \). The component workload \( \mathcal{W} \) is comprised of regular mixed-criticality tasks as well as interface tasks representing the child components. The tasks in the workload \( \mathcal{W} \) are scheduled by the local scheduler \( S \), independently of all the other components in the system.
Component interfaces have been widely used in traditional hierarchical systems to abstractly represent the resource demand and supply of components (see for example [20]). From the component’s perspective, its interface represents the resource demand of its workload. While from the perspective of its parent component or system, the interface represents the resource supply that the parent guarantees. These interfaces of components are essential for satisfying the property of compositionality.

Resource models such as periodic have been previously defined as interfaces for components in traditional hierarchical systems [20]. Analogously, we now present the mixed-criticality periodic resource (MCPR) model for mixed-criticality components. Since we focus on systems with two criticality levels, we assume that the MCPR model can have at most two criticality levels.

Definition 1: A Mixed-Criticality Periodic Resource (MCPR) is defined as $I = (T, L, C)$, where $T$ denotes the period, $L \in \{LC, HC\}$ denotes the criticality level, and $C \subseteq \{C^L, C^H\}$ is a list of resource capacities. $C^L$ denotes LC resource capacity and $C^H$ denotes HC resource capacity.

A component $C$ can be abstracted as an MCPR interface $I = (T, L, C)$, and the corresponding task $T$, $LC$, $HC$ (denoted as interface task) represents $C$ in the workload of its parent component. We assume that period $T$ of this interface is already specified by the system designer as in the standard literature on hierarchical scheduling (e.g., see [20]). For instance, this period could be determined based on either component-level requirements or considerations for overheads such as context-switches. The criticality level $L$ is directly determined by the criticality level of the component it is representing. If $C$ is a $LC$ component, then $L = LC$, otherwise $L = HC$.

Mode of the interface. The semantics of interface $I$ (and the corresponding interface task) depend on its criticality mode at run time, which in turn depends on the criticality mode of component $C$. In fact, we assume that the criticality mode of $I$ is identical to the external mode of $C$. When $C$ experiences EMS, the mode of the interface and interface task switches from $LC$ to $HC$. While the interface is in $LC$ mode, it is guaranteed to request no more than $C^L$ units of resource periodically every $T$ time units from the parent component. But when it switches to $HC$ mode, it can thereafter request up to $C^H$ units of resource periodically.

B. MCPR Supply Bound Function

The supply bound function (sbf) of a resource model characterizes the minimum resource supply guaranteed by the model to the underlying component. In this section, we derive the sbf for a MCPR interface $I = (T, L, C)$ of a component $C = (W, TL, S)$. We let $sbf(I, t)$ denote the sbf for a time interval of length $t$, where $t \leq t$ denotes the time instant for EMS of component $C$. As the resource is supplied periodically, component $C$ is guaranteed to receive either $C^L$ or $C^H$ units of resource every $T$ time units in $LC$ or $HC$ mode, respectively. We use the following additional notations in this section.

- $S$ denotes the start time of the first interface period within time interval $[0, t]$.
- $E$ denotes the end of the interface period.
- $n$ denotes the number of interface periods within interval $[0, t]$.
- $n_E$ denotes the number of interface periods within interval $[0, t_E]$.
- $s_E$ denotes the start of a periodic interface period at time $t_E$, i.e., $s_E \leq t_E < e_E$, where $e_E = s_E + T$.
- $e$ denotes the start of interface period after $t$, i.e., $e = s_1 + n \times T + T$.
- For simplicity of presentation, we also use the short-cut notation $x|_0 = \max\{0, x\}$.

When $t_E = t$, there is no external mode switch for component $C$ in the interval of interest, and the component and interface are only executing in $LC$ mode. Therefore, $sbf(I, t)$ in this case is identical to the sbf defined for periodic resource models with $I$ supplying $C^L$ units of resource periodically [20]. Thus, in this case, minimal resource is supplied when $s_1 = T - C^L$ and $n = \left\lfloor \frac{t - (T - C^L)}{T} \right\rfloor$. We record this sbf in the following equation.

$$sbf(I, t) = n \times C^L + \left[\frac{t - (T - C^L)}{T}\right]_0$$

For the case when $t_E < t$, there are two possible resource supply patterns, denoted A and B, that can lead to the minimum resource supply. We now present these two patterns and the corresponding sbf equations, $sbf(I, t_E, t)|_A$ and $sbf(I, t_E, t)|_B$.

**Pattern A:** $S = T - C^L$. The scenario of pattern A is shown in Figure 3, where $n_E = \left\lfloor \frac{t - (T - C^L)}{T} \right\rfloor$. In the first period, $C^L$ units of resource are supplied as early as possible and hence during $[0, 2 \times (T - C^L)]$, no resource is supplied. In the following periods until time instant $s_E = n_E \times T + T - C^L$, $C^L$ units are supplied as late as possible. In the period $[s_E, e_E]$, the amount of supply depends on the distance of $t_E$ from $s_E$. If $t_E - s_E < C^L$, then the resource supply in this period cannot be exhausted when component $C$ has EMS at $t_E$. Therefore interface $I$ will provide $C^H$ units of resource in this period, because it can signal its mode switch to the parent component. On the other hand, if $t_E - s_E \geq C^L$ then in the example figure, then the resource supply in $[s_E, e_E]$ can be exhausted before component
An important boundary case to consider is when guaranteed to provide $C_L$ units in this period. After time instant $e_E$, the interface is guaranteed to provide $C_H$ units of resource in every period. An important boundary case to consider is when $e = n \times T + 2T - C_L = e_E$ and $t_E - s_E \geq C_L$. That is, when $t_E$ and $t$ are in the same period and the interface can exhaust its resource supply before EMS of component $C$ (scenario shown in Figure 4). In this case, the minimum supply in this period can happen when it is provided as late as possible (for instance when $e - t > C_H - C_L$). We record the sbf corresponding to the pattern of Figures 3 and 4 below.

$$
sbf_i(t_E, t)_{[A]} = \begin{cases} 
& n_E \times C_L + (n - n_E) \times C_H + [t - (2T - C_L - C_H) - n \times T]_0, \quad t_E - s_E < C_L \\
& (n_E + 1) \times C_L + (n - n_E - 1) \times C_H + [t - (2T - C_L - C_H) - n \times T]_0, \quad e \neq e_E, e = e_E, \quad t_E - s_E \geq C_L \\
& n_E \times C_L + \min \{C_L, [t - (2T - C_L - C_H) - n \times T]_0\}, \quad e = e_E, \quad t_E - s_E \geq C_L 
\end{cases}
$$

**Pattern B:** $s_1 = T - C_L - (x_E - t_E)$, where $x_E = \left\lceil \frac{e_E}{T} \right\rceil \times T$. Scenario of pattern B is shown in Figure 5 which is obtained by shifting pattern A in Figure 4 by $x_E - t_E$. In this case,

$$
n_E = \left\lceil \frac{t_E - s_1}{T} \right\rceil, n = \left\lceil \frac{t - s_1}{T} \right\rceil, \quad t_E - s_E = T - C_L + T
$$

and $e = n \times T + T + s_1$.

The sbf corresponding to this shifted supply pattern is given below. It is similar to the previous case, except that the interface period containing $t_E$ is now guaranteed to supply no more than $C_L$ time units.

$$
sbf_i(t_E, t)_{[B]} = \begin{cases} 
& (n_E + 1) \times C_L + (n - n_E - 1) \times C_H + [t - s_1 - (T - C_H) - n \times T]_0, \quad e \neq e_E \\
& n_E \times C_L + \min \{C_L, [t - s_1 - (T - C_H) - n \times T]_0\}, \quad e = e_E
\end{cases}
$$

The following lemma proves that it is sufficient to consider the above two supply patterns for determining the sbf.

**Lemma 6:** When $t_E < t$, pattern A or B are the only two possible supply patterns that can result in the minimal resource supply from interface $I$.

**Proof:** Suppose there exists a $s_1 \in [0, T)$ such that $s_1 \neq T - C_L$ (pattern A) and $s_1 \neq T - C_L - (x_E - t_E)$ (pattern B), but $s_1$ leads to the minimal supply pattern for time interval length $t$. Case 1 ($s_1 = T - C_L + \epsilon 0 < \epsilon \leq C_L$): In this case, it is easy to see that the supply will be greater than or equal to sbf$_i(t_E, t)_{[A]}$, because the supply for the first interface period will increase by $\epsilon$ and the supply for the last interface period will decrease by at most $\epsilon$. Case 2 ($s_1 = T - C_L - (x_E - t_E) + \epsilon 0 < \epsilon < (x_E - t_E)$): In this case, the supply for the interface period containing $t_E$ will stay the same or increase by $C_H - C_L$ while the supply for the last interface period may decrease by at most $\epsilon$ compared with the case when $s_1 = T - C_L - (x_E - t_E)$. Therefore this supply is also minimized when $\epsilon \rightarrow x_E - t_E$ or $\epsilon \rightarrow 0$. Case 3 ($s_1 = T - C_L - (x_E - t_E) - \epsilon 0 < \epsilon \leq T - C_L - (x_E - t_E)$): In this case, the supply for the interface period containing $t_E$ will stay the same, while the supply for the last interface period may stay the same or increase compared with the case when $s_1 = T - C_L - (x_E - t_E)$. Therefore in this case as well, the supply is minimized when $\epsilon \rightarrow 0$. Combining the above cases, we can conclude that the supply is minimized with either pattern A or pattern B.

Thus, a safe lower bound for sbf$_i$ for the case when $t_E < t$ can be stated as follows.

$$
sbf_i(t_E, t) = \min \{sbf_i(t_E, t)_{[A]}, sbf_i(t_E, t)_{[B]}\}
$$

**C. Interface Generation**

In this section we use the sbf, together with the dbf of component $C$, to generate interface $I$. For component $C$ to be schedulable using interface $I$, it is sufficient to ensure that $\text{dbf}(C, t, t_E, t_I) \leq \text{sbf}_i(t_E, t)$ for various time interval lengths. Below we first present the schedulability test for the case when component $C$ does not experience EMS. That is, the interface only executes in $LC$ mode supplying $C_L$ resource capacity periodically.

**Theorem 2:** A mixed-criticality component $C$ is schedulable in $LC$ mode with $sbf_i(t_E, t)$ if, $\forall t : 0 \leq t \leq t_{MAX}, \forall t_I : 0 \leq t_I \leq t$,

$$
\text{dbf}(C, t, t_E, t_I) \leq \text{sbf}_i(t_E, t) \quad \text{If } t_E = t
$$

where $t_{MAX}$ is a pseudo-polynomial in the size of the input that can be derived using similar techniques in Section B in the appendix, and $sbf_i(t_E, t)$ is given by Equation (10) in Section [IV, B].

For a given $t$ and $t_I$, $\text{dbf}(C, t, t_E, t_I)$ can be computed using techniques described in Section [III, C]. The only unknown quantity in Equation (14) is the $LC$ resource capacity $C_L$. This capacity can then be computed exactly using existing techniques [21].

To compute the $HC$ resource capacity $C_H$, we need to consider the schedulability test when component $C$ experiences EMS at some time instant $t_E(< t)$. The following theorem presents this test.

**Theorem 3:** A mixed-criticality component $C$ is schedulable in $HC$ mode with $sbf_i(t_E, t)$ if $\forall t : 0 \leq t \leq t_{MAX}, \forall t_E : 0 \leq t_E \leq t, \forall t_I : 0 \leq t_I \leq t_E$,

$$
\text{dbf}(C, t, t_E, t_I) \leq \text{sbf}_i(t_E, t)
$$

Fig. 5. MCPR worst-case resource supply pattern B
where $sbt_i(t_E, t)$ is given by Equation (15) in Section V-B.

The only unknown quantity in Equation (15) is the HC resource capacity $C^H$, assuming we have already computed $C^L$ using Theorem 2. $C^H$ can then be computed similar to $C^L$ using existing techniques [21].

V. Evaluation

In this section we evaluate the performance of the proposed mechanism in terms of offline schedulability as well as its ability to support LC task executions online. Tasks are generated using the following settings, where each parameter is randomly drawn from the given range using an uniform distribution.

- $u_i^L = C_i^L/T_i$ is in the range $[0.02, 0.1]$.
- $C^H_i/C^L_i$ is in the range $[2, 3]$.
- $T_i$ is in the range $[10, 150]$.
- $D_i = T_i$ as service adaption strategy, one of the mechanisms being compared, can only support implicit deadline tasks.
- Task $\tau_i$ is deemed to be a HC task with probability 0.5.
- For a HC task $\tau_i$, $D_i^L$ is determined by the deadline tuning algorithm in [3].
- For the proposed mechanism, we assume that all the $|H|$ HC tasks in the generated taskset are allocated to a HC component $C_H = \{W_H, TL_H\}$, and all the LC tasks are allocated to a LC component $C_L$.

We have chosen relatively small values for $u_i^L$ and $C^H_i/C^L_i$ so that sufficient number of HC tasks are generated. This enables us to evaluate the online performance of various approaches when different number of HC tasks synchronously switch to HC mode. The generated taskset is evaluated for offline schedulability as well as online performance in terms of support for LC execution under four different mechanisms. These include the mechanism presented in this paper (“Proposed Mechanism”), service adaptation strategy [12] (“Service Adaptation”), Interference Constraint Graph [14] (“ICG”), and the classical mixed-criticality studies in which all the LC jobs are dropped at the moment any HC job switches to HC mode [8] (“Classical Model”). Note that the classical model can be obtained by setting $TL_H = 0$ in our mechanism. In Section V-A we present our results for offline performance based on schedulability tests, and in Section V-B we compare their online performance through simulations.

A. Offline Schedulability

In order to generate feasible tasksets, we consider different bounds for the term $\max\{U_L^L + U_L^H, U_H^H\}$, where $U_L^L = \sum_{L_j=LC} C_{L_j}/T_j$, $U_L^H = \sum_{L_j=HC} C_{L_j}/T_j$, and $U_H^H = \sum_{H_j=HC} C_{H_j}/T_j$. For each bound value, we generate 1000 tasksets based on the procedure described above, and evaluate their off-line schedulability. For the elastic model [11] in which the LC task periods are extended, any generated taskset with $U_H^H$ is always schedulable, because in the worst-case all the LC task periods can be extended to infinity. The schedulability test for the service adaption strategy [6] is a utilization based test. ICG uses the well known Audsley’s algorithm to assign task priorities, and its schedulability is maximized when the interference graph is fully connected, i.e., each HC task has an execution dependency with every LC task in the system. For our mechanism, if a hierarchical scheduling framework is considered, then we assume that the MCPR interface period $T$ for both $C_H$ and $C_L$ is equal to 5 time units. This is reasonable because the smallest task period in any taskset is 10 time units.

Figures 6 and 7 show the schedulability performance for the tasksets under various mechanisms. In Figures 6 we present results for our mechanism under a flat scheduling framework, and in In Figures 6 we present results for our mechanism under a hierarchical scheduling framework. In these figures, the x-axis denotes the bound value for $\max\{U_L^L + U_L^H, U_H^H\}$, and the y-axis denotes schedulability ratio, i.e., percentage of tasksets deemed schedulable by the different mechanisms. For our mechanism, we generate the schedulability results for various values of the tolerance limit: $TL_H = 0, [0.2|H|], [0.4|H|], [0.6|H|], [0.8|H|]$ and $|H|$.

As shown in Figure 6 the schedulability performance of our mechanism clearly depends on the tolerance limit; a higher limit generally implies lower schedulability, because it uses additional resources to support LC executions. For values of $TL_H$ up to $[0.4|H|]$, our mechanism outperforms both service adaptation and ICG on an average. Similar trends can also be observed for our mechanism under a hierarchical framework, except that the schedulability drops more rapidly due to the overhead of hierarchical scheduling. The classical model is represented by the curve with $TL_H = 0$ and it has the highest schedulability, but offers no support for LC executions when $HC$ tasks switch to HC mode. Thus we can conclude that as long as no more than $[0.4|H|]$ of the HC tasks execute in HC mode at each time instant, our mechanism offers the best performance in terms of offline schedulability as well as online support for LC executions.

B. Online Support for LC Executions

In this section, we compare the performance of our mechanism in terms of its ability to support LC executions with the other mechanisms described above. We use the following quantitative parameter to measure this online LC performance.

Definition 2 (Percentage of Finished LC Jobs (PFJ)): Let $MAX_t$ denote the maximum possible number of LC jobs that a taskset $T$ can generate in the time interval $[0, t]$. By definition, $MAX_t = \sum_{t_i=LC} \lfloor t/t_i \rfloor$. Let $FIN_t$ denote the number of LC jobs that successfully finish by their deadlines in the time interval $[0, t]$ using some mechanism. Then, $PFJ$ is equal to $FIN_t/MAX_t$.

Tasksets are generated using the procedure described earlier, and the various mechanisms are simulated to measure their online performance. The following additional settings and restrictions are used for this purpose.

- $\max\{U_L^L + U_L^H, U_H^H\} = 0.8, 0.85$ and 0.9.
- Tolerance limit $TL_H$ is chosen to be the largest value that still guarantees schedulability of our mechanism under a flat scheduling framework.
- Tasksets are simulated for $t = 10,000$ time units.
We have chosen a relatively high value for $\max\{U_L^L + U_H^L, U_H^H\}$, because at smaller values there is sufficient spare capacity so that all the mechanisms are easily able to support LC executions. Simulation results are shown in Figures 8 and 9. The x-axis denotes the probability that a HC job independently switches to HC mode, and the y-axis denotes PFJ for each mechanism. Each point in these figures is generated by taking an average value of PFJ over 1000 tasksets. In Figure 8 we consider only those tasksets that are deemed to be offline schedulable by all the presented mechanisms. As shown in the figure, our mechanism consistently outperforms all the other mechanisms for different values of mode switch probability, and the performance gap improves with increasing probability values. One should note that the results in Figure 8 may not be truly representative of the performance of our mechanism in terms of its ability to support LC jobs, and this can be explained as follows. To compare our mechanism’s ability to support LC executions with the other mechanisms, we have to simulate using tasksets that are schedulable by all these mechanisms. In particular, it does not include many tasksets that are schedulable under our mechanism, but not under one of the other mechanisms. From our observation, in the tasksets that are schedulable by all these mechanisms, the average percentage of HC tasks is much higher than that of LC tasks. Hence to show the ability of our mechanism to support LC executions in a more objective way, we compare the proposed mechanism alone with the classical model, with utilization bound $\max\{U_L^L + U_H^L, U_H^H\} = 0.8, 0.85$ and $0.9$ as shown in Figure 9. In this case, any taskset schedulable by the classical model can be used in the simulation. It can be seen that the performance of both our mechanism and the classical model drops when compared with the results in Figure 8. However, it can also been seen that, our mechanism still dominates the classical model and the corresponding performance gap does not decrease compared with the gap in Figure 8.

**ACKNOWLEDGMENT**

This work was supported in part by MoE Tier-2 grant (MOE2013-T2-2-029) and NTU start-up grant, Singapore. This work was also supported in part by MSIP/IITP (14-824-09-0013) funded by the Korea Government.

**VI. CONCLUSIONS**

In this paper we proposed a novel mechanism to improve the service levels of low-criticality tasks by allowing them to execute even when some high-criticality tasks have exceeded their estimated WCETs. We developed schedulability tests for our mechanism under the mixed-criticality EDF scheduling strategy, considering both a flat as well as an hierarchical scheduling framework. We also evaluated the performance of our mechanism in terms of offline schedulability and online support for low-criticality executions. Simulation results clearly show that the proposed mechanism outperforms all the existing approaches.

In the evaluation section we only consider the performance of our mechanism when all the high-criticality tasks are in one component and all the low-criticality tasks are in another component. In fact, its performance can be further improved if we also consider scenarios in which the low-criticality tasks are allocated to the same component as the high-criticality ones, especially in terms of offline schedulability. In our future work...
we will consider this problem of optimally allocating the low-criticality tasks so as to maximize offline schedulability as well as online performance.

REFERENCES

[1] P. J. Prisaznuk, “Integrated modular avionics,” in Aerospace and Electronics Conference (NAECON). IEEE, 1992, pp. 39–45.

[2] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International. IEEE, 2007, pp. 239–243.

[3] S. Baruah, A. Burns, and R. Davis, “Response-Time Analysis for Mixed-Criticality Systems,” in RTSS, 2011, pp. 34–43.

[4] S. Baruah and G. Fohler, “Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems,” RTSS, pp. 3–12, 2011.

[5] N. Guan, P. Ekberg, M. Stigge, and W. Yi, “Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems,” in RTSS, 2011, pp. 13–23.

[6] S. Baruah, V. Bonifaci, G. D’Angelo, H. Li, and A. Marchetti-Spaccamela, “The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems,” in ECRTS, 2012.

[7] P. Ekberg and W. Yi, “Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks,” in ECRTS, 2012, pp. 135–144.

[8] A. Easwaran, “Demand-based Scheduling of Mixed-Criticality Sporadic Tasks on One Processor,” in RTSS, 2013, pp. 78–87.

[9] A. Burns and S. Baruah, “Towards a More Practical Model for Mixed-Criticality Systems,” in Workshop on Mixed-Criticality Systems (co-located with RTSS), 2013.

[10] M. Jan, L. Zaourar, and M. Pitel, “Maximizing the execution rate of low-criticality tasks in mixed criticality system,” in Workshop on Mixed-Criticality Systems (co-located with RTSS), 2013.

[11] H. Su and D. Zhu, “An elastic mixed-criticality task model and its scheduling algorithm,” in Proceedings of the Conference on Design, Automation and Test in Europe, ser. DATE ’13. San Jose, CA, USA: EDA Consortium, 2013, pp. 147–152. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485325

[12] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, “Service adaptions for mixed-criticality systems,” in In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

[13] A. Lackorzynska, A. War, M. Volp, and H. Hartig, “Flattening hierarchical scheduling,” in Proceedings of the Tenth ACM International Conference on Embedded Software (EMSOFT), 2012, pp. 93–102.

[14] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, “Interference constraint grapha new specification for mixed-criticality systems;” in Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on. IEEE, 2013, pp. 1–8.

[15] T. Fieninger and A. Burns, “Incorporating the notion of importance into mixed criticality systems,” in WMC, 2014, p. 33.

[16] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson, “RTOS support for multi-core mixed-criticality systems,” in RTAS, 2012.

[17] “Deos: A Time & Space Partitioned DO-178 Level A Certifiable RTOS,” http://www.ddci.com/products_deos.html.

[18] “ARINC653 - An Avionics Standard for Safe, Partitioned Systems”. Wind River Systems / IEEE Seminar, 2008.

[19] S. Baruah, A. Mok, and L. Rosier, “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor,” in RTSS, 1990, pp. 182–190.

[20] I. Shin and I. Lee, “Periodic resource model for compositional real-time guarantees,” in RTSS, 2003, pp. 2–13.

[21] A. Easwaran, M. Anand, and I. Lee, “Compositional Analysis Framework using EDP Resource Models,” in RTSS, 2007, pp. 129–138.

APPENDIX

A. Dbf Optimization

When component C experiences EMS, i.e., the case when $t_E < t$, it is pessimistic to simply add up the demand of all the tasks. Here we introduce an optimization that can be applied in the schedulability test to reduce this pessimism. We split $dbf(\tau_i, t, t_j)$ into two elements, $DL(\tau_i, t, t_j)$ denoting the demand for the interval $[0, t_E)$, and $DH(\tau_i, t, t_j)$ denoting the demand for the interval $[t_E, t]$.

$$dbf(\tau_i, t, t_j) = DL(\tau_i, t, t_j) + DH(\tau_i, t, t_j) \quad (16)$$

Below we present a key observation that provides some insight into this split. Since the first deadline miss is assumed to happen at time instant $t$ in our schedulability test, the demand before $t_E < t$ cannot exceed $t_E$. Otherwise, the first deadline miss would happen at or before $t_E$. Thus the total demand during $[0, t_E)$ can be bounded by $t_E$, and as a consequence $dbf(C, t, t_E, t_I)$ can be more tightly bounded as follows.

$$dbf(C, t, t_E, t_I) = DL + DH + \sum_{\Delta_t \in \mathcal{G}} \Delta_t$$

$$DH = \sum_{L_i = LC} DH(\tau_i, t, t_j) + \sum_{L_i = HC} DH(\tau_i, t, t_E),$$

$$DL = \min \left\{ t_E, \sum_{L_i = LC} DL(\tau_i, t, t_j) + \sum_{L_i = HC} DL(\tau_i, t, t_E) \right\} \quad (17)$$

In Equation [17] we use DL to bound the total demand of C for the interval $[0, t_E)$, and DH to bound the total demand for the interval $[t_E, t)$. In order to maximize the total demand, we must then split the demand between DL and DH such that DH is maximized (or equivalently DL is minimized). This is because the total demand for the interval $[0, t_E)$ is bounded by $t_E$.

In Section [11-12] we already present $dbf(\tau_i, t, t_j)$ when task $\tau_i$ satisfies condition a, b, c or d. Here we present $dbf(\tau_i, t, t_j)$ and $DH(\tau_i, t, t_j)$ for these cases, such that DH is maximized. If $\tau_i$ is a LC task, then it cannot execute after $t_E$ (dropped at $t_i = t_E$). Hence for condition a,

$$DL(\tau_i, t, t_j)_{[a]} = dbf(\tau_i, t, t_j)_{[a]}$$

$$DH(\tau_i, t, t_j)_{[a]} = 0 \quad (18)$$

Consider the case when $\tau_i$ satisfies condition b, i.e., $L_i = HC$ and $t - t_i < D_i$. Here as well $\tau_i$ cannot execute after $t_E$ as given in Lemma 2. Hence,

$$DL(\tau_i, t, t_E)_{[b]} = dbf(\tau_i, t, t_E)_{[b]}$$

$$DH(\tau_i, t, t_E)_{[b]} = 0 \quad (19)$$

![Fig. 10. DL(τ_i, t, t_E)_{[c]} and DH(τ_i, t, t_E)_{[c]}](image)
To minimize the demand of $J^A$ before $t_E$, we assume that it executes as late as possible. Thus, $J^A$’s demand before $t_E$ can be bounded by $t_E - r(J^A) - (D^L_i - C^L_i)$, and we have,

$$DL(\tau, t, t_E) = \min \left\{ \left[ t_E - r(J^A) - (D^L_i - C^L_i) \right]_0, C^L_i \right\} + b_i \times C^L_i$$

$$DH(\tau, t, t_E) = -\min \left\{ \left[ t_E - r(J^A) - (D^L_i - C^L_i) \right]_0, C^L_i \right\} + b_i \times C^L_i$$

Finally, consider the case when $\tau_i$ satisfies condition d.

Consider a mixed-criticality system with $p$ HC components $C_1, C_2, \ldots, C_p$ and $q$ LC components $C_{p+1}, C_{p+2}, \ldots, C_{p+q}$. Let $U^L(\cdot)$ and $U^H(\cdot)$ be the utility functions for the LC and HC nodes, respectively.

**Case 1:** If component $C_j$ experiences IMS at $t_{i,j}$, then the demand of a LC task $\tau$ in the time interval $[0, t]$ is upper bounded by $(t_{i,j}/T_i + 1) \times C^L_i$, because $\tau_i$ will be dropped after $t_{i,j}$.

A HC task $\tau_i$ in $C_j$ switches to HC mode at some time instant $t_i \in [t_{i,j}, t_E]$. The demand of $\tau_i$ before $t_E$ is upper bounded by $t_{i,j}/T_i \times C^L_i$, the demand of job $J^A$ is bounded by $C^H_i$, and the demand of task $t_i$ is bounded by $(t - t_i - D_i + T_i)/T_i \times C^H_i$. Thus the total demand of $\tau_i$ in the time interval $[0, t]$ is bounded by

$$t \times C^L_i + C^H_i + \frac{t - t_i - D_i + T_i}{T_i} \times C^H_i$$

Since $C^H_i > C^L_i$ and $t_i \in [t_{i,j}, t_E]$, the value of Expression (22) is maximized when $t_i = t_{i,j}$. Therefore the total demand of $C_j$ is bounded by

$$\sum_{\tau_i \in C_j} \left( \frac{t_{i,j} \times C^L_i + C^H_i (t - t_{i,j} - D_i + 2T_i)}{T_i} \right) + \sum \frac{(t_{i,j}/T_i + 1) \times C^L_i}{T_i}$$

$$\leq U^H(\cdot) + U^L(\cdot) + U^H(\cdot) \times t_{i,j}$$

**Case 2:** Suppose component $C_j$ does not experience IMS, i.e., all the LC tasks within $C_j$ are dropped after $t_E$, and all the HC tasks switch to HC mode at $t_E$. In this case, the demand of a LC task $\tau_i$ in the time interval $[0, t]$ is upper bounded by $(t_E/T_i + 1) \times C^L_i$, and the demand of a HC task $\tau_i$ in the time interval $[0, t]$ is upper bounded by $\frac{t_E}{T_i} \times C^L_i + C^H + \frac{t_{i,j} - t_{i,j} - D_i + T_i}{T_i} \times C^H$. Therefore the total demand of $C_j$ is bounded by

$$\sum_{\tau_i \in C_j} \left( t_E \times C^L_i + C^H \times (t - t_E - D_i + T_i) \right) / T_i$$

$$+ \sum_{\tau_i \in C_j} \left( t_E \times C^L_i + C^H \times (t - t_E - D_i + T_i) \right) / T_i$$

$$\leq U^H(\cdot) + U^L(\cdot) + U^H(\cdot) \times t_{i,j}$$

Let $A$ denote the set of components $C_j$ with $U^L(j) + U^H(j) - U^H(j) < 0$, and $B$ denote the remaining set of components. Then if $C_j \in A$, its demand bound given above is maximized when $t_{i,j} = 0$ or $t_E = 0$. On the other hand, if $C_j \in B$, its demand bound is maximized when $t_{i,j} = t_E - t$. Thus, an upper bound on the total demand of $C_j$ is equal to

$$\sum_{\tau_i \in C_j} \left( \frac{t_E \times C^L_i + C^H \times (t - t_E - D_i + T_i)}{T_i} \right) / T_i$$

$$+ \sum_{\tau_i \in C_j} \left( \frac{t_E \times C^L_i + C^H \times (t - t_E - D_i + T_i)}{T_i} \right) / T_i$$

$$\leq U^H(\cdot) + U^L(\cdot) + U^H(\cdot) \times t_{i,j}$$

Suppose $\sum_{j=1}^{j \leq p+q} \text{dbf}(C_j, t, t_E, t_{i,j}) > t$ for some $t$. Then it must be the case that

$$\sum_{j=1}^{j \leq p+q} \left( \max_{i \in A} \left( 2T_i - D_i \right) \times U^H(j) \right) \times \frac{t_{i,j}}{T_i} \times C^L_i$$

$$> t \left( 1 - \sum_{j \in B} \left( U^L(j) + U^H(j) \right) \right)$$

$$\Rightarrow t < \frac{\sum_{j=1}^{j \leq p+q} \left( \max_{i \in A} \left( 2T_i - D_i \right) \times U^H(j) \right) \times \frac{t_{i,j}}{T_i} \times C^L_i}{1 - \sum_{j \in A} \left( U^L(j) + U^H(j) \right)}$$

Thus we can conclude that the upper bound of $t$, i.e., $t_{MAX}$, is given as

$$\sum_{j=1}^{j \leq p+q} \left( \max_{i \in A} \left( 2T_i - D_i \right) \times U^H(j) \right) \times \frac{t_{i,j}}{T_i} \times C^L_i$$

$$1 - \sum_{j \in A} \left( U^L(j) + U^H(j) \right)$$