University of Pennsylvania
ScholarlyCommons

Departmental Papers (CIS) | Department of Computer & Information Science

4-2013

Overhead-Aware Compositional Analysis of Real-Time Systems

Linh T.X. Phan
University of Pennsylvania, linhphan@cis.upenn.edu

Meng Xu
University of Pennsylvania, mengxu@cis.upenn.edu

Jaewoo Lee
University of Pennsylvania, jaewoo@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_papers

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Linh T.X. Phan, Meng Xu, Jaewoo Lee, Insup Lee, and Oleg Sokolsky, "Overhead-Aware Compositional Analysis of Real-Time Systems", *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)* , 237-246. April 2013.
http://dx.doi.org/10.1109/RTAS.2013.6531096

19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS' 13) (as part of the Cyber-Physical Systems Week (CPSWeek)), Philadelphia, PA, April 9-11, 2013.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_papers/789
For more information, please contact libraryrepository@pobox.upenn.edu.

Overhead-Aware Compositional Analysis of Real-Time Systems

Abstract
Over the past decade, interface-based compositional schedulability analysis has emerged as an effective method for guaranteeing real-time properties in complex systems. Several interfaces and interface computation methods have been developed, and they offer a range of tradeoffs between the complexity and the accuracy of the analysis. However, none of the existing methods consider *platform overheads* in the component interfaces. As a result, although the analysis results are sound in theory, the systems may violate their timing constraints when running on realistic platforms. This is due to various overheads, such as task release delays, interrupts, cache effects, and context switches. Simple solutions, such as increasing the interface budget or the tasks' worst-case execution times by a fixed amount, are either unsafe (because of the overhead accumulation problem) or they waste a lot of resources.

In this paper, we present an *overhead-aware* compositional analysis technique that can account for platform overheads in the representation and computation of component interfaces. Our technique extends previous overhead accounting methods, but it additionally addresses the new challenges that are specific to the compositional scheduling setting. To demonstrate that our technique is practical, we report results from an extensive evaluation on a realistic platform.

Keywords
multiprocessing systems, processor scheduling, cache effect, context switches, interface-based compositional schedulability analysis, overhead accounting method, overhead-aware compositional analysis, real-time system, task release delay, worst-case execution times, bandwidth, context, delays, interference, real-time systems, schedules

Disciplines
Computer Engineering | Computer Sciences

Comments
19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS' 13) (as part of the Cyber-Physical Systems Week (CPSWeek)), Philadelphia, PA, April 9-11, 2013.

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/789

Overhead-Aware Compositional Analysis of Real-Time Systems*

Linh T.X. Phan Meng Xu Jaewoo Lee Insup Lee Oleg Sokolsky
Department of Computer and Information Sciences, University of Pennsylvania
Email: {linhphan,mengxu,jaewoo,lee,sokolsky}@cis.upenn.edu

***Abstract*—Over the past decade, interface-based compositional schedulability analysis has emerged as an effective method for guaranteeing real-time properties in complex systems. Several interfaces and interface computation methods have been developed, and they offer a range of tradeoffs between the complexity and the accuracy of the analysis. However, none of the existing methods consider *platform overheads* in the component interfaces. As a result, although the analysis results are sound in theory, the systems may violate their timing constraints when running on realistic platforms. This is due to various overheads, such as task release delays, interrupts, cache effects, and context switches. Simple solutions, such as increasing the interface budget or the tasks' worst-case execution times by a fixed amount, are either unsafe (because of the overhead accumulation problem) or they waste a lot of resources.**

In this paper, we present an *overhead-aware* compositional analysis technique that can account for platform overheads in the representation and computation of component interfaces. Our technique extends previous overhead accounting methods, but it additionally addresses the new challenges that are specific to the compositional scheduling setting. To demonstrate that our technique is practical, we report results from an extensive evaluation on a realistic platform.

I. Introduction

The growing complexity of modern real-time systems has brought forward two major trends. First, it is becoming increasingly common to run multiple systems on a shared computing platform, rather than deploying them separately on different physical processors. Second, complex systems are increasingly being created by integrating smaller subsystems that were developed independently. While these current trends help reduce cost and development efforts, they also add many challenges to the timing analysis of such systems.

One effective way to tackle these challenges is to use a *compositional schedulability analysis (CSA)* [28]. In a CSA framework, the system is partitioned into a tree of components that are scheduled hierarchically [29]. The schedulability analysis of such a system is done by constructing a *resource interface* for each component, which abstracts away the detailed timing information of the tasks and exposes only the minimum resources required to satisfy the component's resource demands. The system as a whole is schedulable if the resource requirements in its top-level interface can be satisfied. Thus, CSA techniques enable the composition and isolation of independently developed real-time systems while preserving their timing guarantees.

CSA was introduced on top of hierarchical scheduling [17], [23], [29] nearly a decade ago by Shin and Lee [28], and it has since received considerable attention (see e.g., [4], [13], [15], [22], [24], [27]). Many aspects of CSA are well-understood; for instance, prior work has developed a number of interface computation methods and representations for resource interfaces [4], [13], [15], [18], as well as resource sharing protocols for compositional scheduling (e.g., [7]). However, to the best of our knowledge, all existing CSA theories assume a somewhat idealized platform in which various overheads – such as release delays, preemption overheads, cache effects, and context switches – are negligible.

In practice, assuming that platform overheads are negligible is not realistic: release delays, preemption overheads, cache effects and context switches can significantly interfere with the execution of the tasks. Without taking such overheads into account, the computed interfaces can underestimate the amount of resources required to feasibly schedule the tasks within the underlying components; as a result, tasks can miss their deadlines even if their interfaces are satisfied. In other words, even if a system is schedulable in theory, it can still violate its timing constraints on a realistic platform.

In this paper, we extend the existing compositional schedulability analysis with *overhead-aware interfaces*, and we present the corresponding interface computation methods for a uniprocessor platform. Such an overhead-aware compositional analysis is crucial to bridging the current gap between theory and practice of compositional scheduling theory.

An overhead-aware interface analysis must address two key challenges. First, certain types of overheads cannot be quantified as part of the tasks' worst-case execution time (WCET) because these overheads have no notion of deadlines and cannot be scheduled. For example, interrupts and task release events need to be served by the operating system as soon as they arrive, rather than being scheduled together with the tasks [8], [11]. As a result, existing deadline-sensitive resource interfaces, such as the periodic resource model (PRM) [28] and the explicit deadline periodic model (EDP) [15] cannot be used.

Second, due to the *overhead accumulation* problem, the platform overheads experienced by a component cannot be safely bounded by a fixed amount of resources. For instance, the scheduling of a component may be delayed by interrupt processing or by release events that arrive in another component. Hence, the platform overhead that a component can experience increases with the number of other tasks (components) in the system. However, in a compositional analysis setting, a component's task-level details are not usually available to other components, so it is not possible to compute a safe bound on the resource overhead originating from other components.

Our approach is to identify methods for accounting the two categories of overheads and then, extend existing interfaces

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

to encapsulate the overheads that cannot be included in the tasks' WCETs. The first category consists of all overheads that can be included as part of the tasks' WCETs, such as cache effects and context switches; these can be captured together with tasks' resource demands by a traditional resource model, such as PRM or EDP. The second category consists of the overheads that *cannot* be included as part of the tasks' WCETs, such as interrupt overheads and release delays; these are captured using the deadline-insensitive request bound function [25]. By separating these two categories, our interface representation not only leverages the existing compositional analysis results but also enables the computation of inter-component overheads during the interface composition; thus, we can overcome the overhead accumulation problem.

Contributions. This paper makes the following contributions:

- We discuss the overhead accumulation problem and the insufficiency of the WCET inflation method (Section III).
- We present a compositional overhead account method (Section IV) and introduce an overhead-aware schedulability test for components (Section V).
- We propose an overhead-aware representation for component interfaces and the corresponding interface computation method (Section VI).
- We illustrate the applicability and benefits of overhead-aware analysis with an extensive evaluation of randomly generated workloads on a realistic platform. The evaluation results show that the computed interfaces are safe in practice, and that they can help reduce resource bandwidth up to a factor of five compared to a baseline approach which inflates tasks' WCETs. (Section VII).

II. Background

This section describes the system model and the background required for our analysis in the coming sections.

System model. The system consists of a tree of components that are scheduled hierarchically. Each internal vertex of the tree represents a *composite component*, whose children are its sub-components. Each leaf represents an *elementary component*, which is a finite set of tasks in the system. Each component has its own scheduler, such as Earliest Deadline First (EDF) or Rate Monotonic (RM), that schedules its sub-components (tasks). All tasks are periodic tasks with explicit hard deadlines. Each task τ_i is defined by $\tau_i = (p_i, e_i, d_i)$, where p_i is the period, e_i is the WCET, d_i is the relative deadline, and $0 < e_i \leq d_i \leq p_i$.

Resource models. We use the explicit deadline resource model (EDP) for part of our interface representation because it is simple and offers a good tradeoff between accuracy and efficiency. (However, our method can easily be incorporated into other resource models in a similar fashion.) An EDP is defined by $\Gamma = (\Pi, \Theta, \Delta)$, where Π is the period, Θ is the budget, and Δ is the deadline. This represents a resource supply that provides Θ units of resources within Δ time units, with this pattern repeating every Π time units. The *bandwidth* of the EDP is defined by Θ/Π. We say that an EDP is (bandwidth) *optimal* for a component iff it has the smallest bandwidth among all EDPs that can feasibly schedule the tasks within the component.

The amount of the resource provided by a resource model R is captured by the supply bound function of R, denoted by $\mathsf{sbf}_R(t)$, which specifies the minimum number of resource units that R is guaranteed to provide over any time interval of length t, for all $t \geq 0$. The SBF of an EDP is given by [15]:

$$\mathsf{sbf}_\Gamma(t) = \begin{cases} y\Theta + \max\{0,\, t - x - y\Pi\}, & \text{if } t \geq \Delta - \Theta \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $x = \Pi + \Delta - 2\Theta$ and $y = \lfloor \frac{t-(\Delta-\Theta)}{\Pi} \rfloor$. We call x the *blackout interval* of Γ. We note that an SBF can also be used as a resource interface; in which case, the supply bound function of an SBF is the SBF itself.

Schedulability analysis without platform overheads. Let C be a component with a workload $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where $\tau_i = (p_i, e_i, d_i)$ is either a periodic task or an EDP interface of a subcomponent of C. The request bound function (RBF) of τ_i is given by $\mathsf{rbf}_{\tau_i}(t) = \lceil \frac{t}{p_i} \rceil e_i$ for all $1 \leq i \leq n$. Under a fixed-priority (FP) scheduling algorithm, such as Rate Monotonic (RM) or Deadline Monotonic (DM), the RBF of the i highest-priority tasks of C is given by [21]:

$$\mathsf{rbf}_{C,i}(t) = \sum_{1 \leq k \leq i} \mathsf{rbf}_{\tau_k}(t),\ \forall 1 \leq i \leq n,$$

where τ_i has higher priority than τ_j if $i < j$. Under EDF, the demand bound function (DBF) of C is given by:

$$\mathsf{dbf}_C(t) = \sum_{i=1}^{n} \left(\left\lfloor \frac{t + p_i - d_i}{p_i} \right\rfloor e_i \right).$$

The next two lemmas state the schedulability condition of C, assuming no resource overheads [15]:

Lemma 1. *A component C is schedulable under an FP scheduling algorithm by a resource model R iff*

$$\forall\ 1 \leq i \leq n,\ \exists t \in [0, d_i]\ s.t.\ \ \mathsf{sbf}_R(t) \geq \mathsf{rbf}_{C,i}(t). \tag{2}$$

Lemma 2. *A component C is schedulable under EDF by a resource model R iff $\mathsf{sbf}_R(t) \geq \mathsf{dbf}_C(t)$ for all $0 < t \leq \mathsf{LCM}$, where LCM is the least common multiple of p_i for all τ_i in C.*

Based on the above schedulability tests, an EDP interface of a component can be computed efficiently using the algorithms proposed in [15]. Due to space constraints, we refer the readers to [15] for the details.

III. Motivating example

Next, we present an example that illustrates the overhead accumulation problem and the issues that can arise with existing compositional analysis methods in practice when platform overheads are neglected. For this example, we focus on task release interference.

Real-time jobs are typically released using timer interrupts (for periodic tasks) or other interrupt routines (for aperiodic tasks) [11]. We use the term *release ISR* to denote the processing of an interrupt service routine that releases a job. On a realistic platform, such release ISRs take non-zero time and need to be serviced by the kernel as soon as they arrive [8], [11].

Example of invalid analysis results due to overhead. Consider a component C_{root} that has two subcomponents, C_1 and C_2, where C_1 has the workload $\{\tau_1\}$ and C_2 has the workload $\{\tau_2, \ldots, \tau_{51}\}$. All components schedule their tasks (subcomponents) under EDF. The timing parameters of the tasks are $\tau_1 = (5, 4, 5)$ and $\tau_2 = \cdots = \tau_{51} = (500, 1, 500)$, where the time unit is milliseconds (ms). Each release ISR takes up to 0.020 ms, which is a typical value in practice [11].

By applying the existing interface computation techniques for EDP (c.f. Section II), we obtain the interfaces $I_1 = (5, 4, 5)$, $I_2 = \{10, 1, 10\}$ and $I_{\mathsf{root}} = \{5, 4.5, 5\}$. Since the bandwidth of I_{root} is 0.9, this system is deemed to be schedulable. However, in practice this system is *not* schedulable because of the scenario shown in Fig. 1.

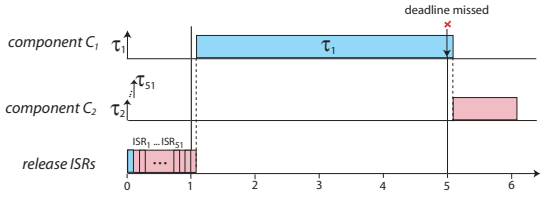


Fig. 1: A counterexample for the schedulability of C_{root}.

In this scenario, the ideal release times of τ_1 and τ_2 are 0, and each τ_i is ideally released 0.001 ms after τ_{i-1} for all $i > 2$. In this case, although τ_1 is effectively released at time 0.020, it has to wait for the system to finish processing all the release ISRs in the second component. As a result, it can only start its execution at time 1.020 and finish at time 5.020. Hence, τ_1 misses its deadline.

Why is WCET inflation not sufficient? Intuitively, it may seem that a simple solution is to inflate the WCETs of each task by the execution time of one release ISR. However, even with inflated WCETs, the above system is still deemed schedulable by the analysis. Inflation is unsafe because it does not account for the release interference by *other* tasks' release ISRs: for instance, the task τ_1 in Fig. 1 is delayed by the release ISRs of all the other tasks in the system. Even if we inflate the task WCET by the total execution time of *all* the release ISRs, we may still underestimate the resource needs because a job of a task τ_i may be delayed by multiple release ISRs of multiple jobs of τ_j if p_i is much larger than p_j. We also observe that the release interference overheads accumulate as the number of tasks in system increases.

It would be safe to inflate the WCET of each task by the total execution time of all release ISRs that can delay that task's execution; however, this cannot be done in the compositional setting. In this setting, the timing information of the tasks within one component is unavailable to the other components; thus, the tasks' WCETs and the components' interfaces would need to be recomputed as more tasks (components) are added into the system. This is not possible with current CSA methods because the interfaces do not contain any task-level details or information about overhead. Recomputing the interfaces of components at all levels of the scheduling hierarchy is not desirable because it increases the complexity of the analysis and diminishes the efficiency benefits of CSA.

IV. Overhead Accounting

In this section, we first identify the different types of platform overheads, and then we show how to overcome the problem that was described in the previous section.

A. Overview

The overheads that a job may experience during its lifetime include the following:

- Release ISR (Δ^{rel}): The maximum time needed to add a released job to the ready queue;
- Schedule function (Δ^{sch}): The maximum time the scheduler needs to select the highest-priority job to execute;
- Context switch (Δ^{cxs}): The maximum time the processor needs to switch from one job to another.
- Cache-related preemption (Δ_i^{CRPD}): The maximum time needed to recover the cache affinity of a job after it has been preempted by a task τ_i. (When a job resumes its execution after being preempted, some or all cache blocks of its working set may have been evicted from cache and may need to be reloaded to recover the cache affinity.)
- Tick ($\Delta^{\mathsf{tick}}, p_{\mathsf{tick}}$): The maximum time needed to execute the tick function, which is invoked with period p_{tick}.
- Other interrupts: Time needed to handle other types of interrupts, e.g., network interrupts.

Types of overheads. We group the overheads that affect a task's response time in a compositional scheduling system into two categories. The first category consists of overheads that cannot be accounted for by inflating the tasks' WCETs. It includes the *release interference overheads*, i.e., the overhead a task experiences due to the execution of release ISRs, and overheads due to other interrupts. In this paper, we consider only the release interference overheads, but other types of interrupts can be accounted for in a similar way, provided that their worst-case execution times and a bound on the number of interrupts are known. The second category consists of the remaining types of overheads, which can be accounted for by inflating the tasks' WCETs. We call them *inflatable* overheads.

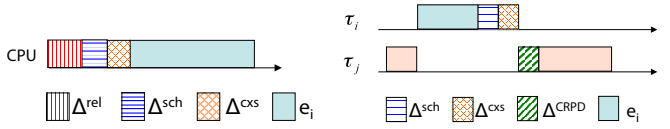
In the following, we describe our method for accounting inflatable overheads. Non-inflatable overhead accounting will be discussed in Section VI.

B. Accounting for inflatable overheads

To account for inflatable overheads, we extend the accounting technique proposed by Brandenburg et al. [11] for use with compositional scheduling. The original method from [11] cannot be applied directly for two reasons: First, it assumes that the number of tasks in the entire system is known a priori; as was discussed earlier, this is not the case in our setting because a component has no knowledge of other components. Second, the original method assumes a constant bound Δ^{CRPD} on the cache-related preemption overhead for each task. This is unsafe in the compositional analysis setting because the cache-related preemption delay of each task depends on its working set size and preemption points, which can change when tasks or components are added to, or removed from, the system.

We extend the method from [11] in two ways: we account for the release interference separately (c.f. Section VI),

rather than simply inflating the WCET, and we introduce a method for quantifying the cache-related preemption overhead (CRPD) of each task, which can be done independently of other tasks. Below, we first describe how inflatable overheads can be quantified based on the events that cause them.



(a) Release event related overhead. (b) Preemption event related overhead.

Fig. 2: Overheads related to release and preemption events.

Overhead related to release events. The timeline of a task's release event is illustrated in Fig. 2(a). τ_i's overhead includes its own release overhead, scheduling overhead, and context-switch overhead. Although it is possible to include τ_i's own release ISRs as part of this overhead, this leads to a pessimistic analysis because we need to include it in the (non-inflatable) release-interference overhead as well. Hence, we only account for the scheduling and the context switch here; thus, the overhead related to release events is:

$$\Delta^{\mathsf{relEv}} = \Delta^{sch} + \Delta^{cxs} \tag{3}$$

This overhead is included once in each job's WCET (since every job is released exactly once).

Overhead related to preemption events. This overhead is illustrated in Fig. 2(b). When a preemption happens, we refer to the higher-priority task τ_i as the *preempting* task and to the lower-priority task τ_j as the *preempted* task. When τ_i finishes its execution, the scheduler is invoked to resume τ_j, which results in a scheduling overhead and a context switch. Further, the system needs to reload any cache blocks of τ_j that have been evicted from the cache during the execution of τ_i, which results in a cache-related preemption overhead. Since a job of τ_i only preempts τ_j exactly once, we include the preemption-related overhead into the WCET of τ_i; the overhead can be computed as:

$$\Delta^{\mathsf{preEv}} = \Delta^{sch} + \Delta^{cxs} + \Delta^{\mathsf{CRPD}_i} \tag{4}$$

The CRPD overhead of τ_i can be quantified based on the evicting cache block (ECB) of τ_i. Here, a memory cache block is called an ECB of τ_i if it can be accessed during the execution of τ_i [1]. Let BRT be the time needed to reload one cache block from the main memory to the cache, and $|\mathsf{ECB}_i|$ be the number of evicting cache blocks of τ_i. Then, the CRPD overhead can be bounded based solely on τ_i as below.

$$\Delta^{\mathsf{CRPD}_i} \leq \mathsf{BRT} \times |\mathsf{ECB}_i| \tag{5}$$

Note that the above bound can be determined separately for each task τ_i, so it can be used within a compositional analysis, where the details of tasks in other components are not known a priori.

Tick overhead. Due to the execution of the tick function, the system loses Δ^{tick} time units every p_{tick} time units. Thus, the effective WCET of each task τ_i with a given WCET e_i, without considering other types of overheads, is given by [9]:

$$e'_i = \lceil \frac{e_i}{p_{\mathsf{tick}} - \Delta^{\mathsf{tick}}} \rceil p_{\mathsf{tick}} \tag{6}$$

Inflated WCET. Based on the above equations, the inflated WCET of a task τ_i can be computed by:

$$e'_i = \lceil \frac{e_i + \Delta^{\mathsf{relEv}} + \Delta^{\mathsf{preEv}}}{p_{\mathsf{tick}} - \Delta^{\mathsf{tick}}} \rceil p_{\mathsf{tick}} \tag{7}$$

V. Overhead-aware Schedulability Analysis of Components

In this section, we extend the existing component schedulability analysis (see Lemmas 1 and 2) to account for the platform overheads a component may experience. We first present the schedulability analysis in the presence of inflatable overheads only, and then we introduce our method for capturing the release interference overheads.

In the following, we consider a component $C = \langle \tau, A \rangle$ with a workload $\tau = \{\tau_1, \ldots, \tau_n\}$ and scheduling algorithm A, where each $\tau_i = (p_i, e_i, d_i,)$ is a periodic task (or a task corresponding to an interface of a subcomponent of C). Let e'_i be the inflated WCET of τ_i, which is computed using Eq. (7). Further, let $\tau' = \{\tau'_1, \ldots, \tau'_n\}$, where $\tau'_i = (p_i, e'_i, d_i)$. We call τ' the *inflated workload* of C. Recall that e'_i includes the overheads due to the inflatable overheads, which include overheads caused by release events, preemption events and ticks (c.f. Section IV).

Lemma 3. ***(Inflatable Overhead-Aware Schedulability Test)*** *A component $C = \langle \tau, A \rangle$ is schedulable by a resource R in presence of inflatable overheads if its inflated workload τ' is schedulable by R under A when there are zero overheads.*

Proof: The lemma is established based on the overhead accounting technique in Section IV. Recall that the inflated WCET, e'_i, is computed based on the maximum values Δ^{relEv}, Δ^{preEv} and Δ^{tick} of the overheads caused by release events, preemption events and ticks for each τ_i, respectively.

Therefore, the total execution time of τ_i in presence of these overheads is no more than e'_i. Hence, we imply that if τ'_i is schedulable under A by the resource R assuming zero overheads, then τ_i is also schedulable under the overheads caused by the release events, preemption events and ticks. In other words, τ is schedulable in presence of inflatable overheads if τ' is schedulable assuming zero overheads. ■

Release interference overhead. Recall that release interference overhead is the delay a task experiences due to the execution of the release ISRs of the tasks in the system.

These release ISRs have the highest priority, and thus they can delay the execution of any task in the system. When multiple release ISRs arrive at the same time, the system executes them one by one in an arbitrary order.[1]

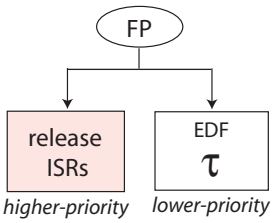


Fig. 3: Modeling release interference overheads.

Our approach is to capture the execution of the release ISRs and the execution of the workload τ of a component C using a compositional scheduling analogy. As is illustrated in Fig. 3, the release ISRs can be seen as the workload within a higher-priority component, and the workload τ forms the lower-priority component. The system's

[1]This is the case for most common platforms [9]. Note that although some systems, e.g., LITMUS [12], execute release ISRs that arrive at the same time in group, the worst-case interference still occur when the release ISRs arrive an epsilon time one after another.

resources are always allocated to the release ISR component before they are given to the workload component. Within the workload component, the tasks are scheduled as usual by C's scheduler, e.g., EDF in Fig. 3.

Based on the above modeling approach, we can determine the total release interference overheads a component experiences due to the release ISRs of its tasks based on the resource requests of these ISRs. We call this the *intra-component* release interference overhead.

Lemma 4. *The intra-component release interference overhead of a workload τ is bounded by the resource request bound function (RBF) of the release ISRs of the tasks in τ, given by*

$$\mathsf{rbf}^{\mathsf{ISR}}_{\tau}(t) = \sum_{\tau_i \in \tau} \mathsf{rbf}^{\mathsf{ISR}}_{\tau_i}(t), \quad \textit{with} \quad \mathsf{rbf}^{\mathsf{ISR}}_{\tau_i}(t) = \lceil \frac{t}{p_i} \rceil \Delta^{rel},$$

where Δ^{rel} is the maximum execution time of a release ISR.

Proof: Since every job release creates one release ISR, the maximum number of release ISRs of task τ_i over any interval of length t is $\lceil \frac{t}{p_i} \rceil$. Since each release ISR requires a maximum of Δ^{rel} execution time units, the total amount of resources requested by the release ISRs of τ_i is $\mathsf{rbf}^{\mathsf{ISR}}_{\tau_i}(t) = \lceil \frac{t}{p_i} \rceil \Delta^{rel}$. As a result, the total amount of resources requested by the release ISRs of all tasks in τ over an interval of length τ is $\mathsf{rbf}^{\mathsf{ISR}}_{\tau}(t) = \sum_{\tau_i \in \tau} \mathsf{rbf}^{\mathsf{ISR}}_{\tau_i}(t)$. Since the release ISRs have higher priority than any other task in the component, $\mathsf{rbf}^{\mathsf{ISR}}_{\tau}(t)$ is also the resource interference that the tasks in τ experience due to their release ISRs. This proves the lemma. ∎

The next two lemmas state the schedulability analysis for a component considering release interference overheads.

Lemma 5. ***(Release Interference-Aware Schedulability Test)*** *A component $C = \langle \tau, A \rangle$ is schedulable by a resource R in presence of intra-component release interference overhead if it is schedulable by a resource R' in the absence of overhead, where R' has a supply bound function equal to*

$$\mathsf{sbf}_{R'}(t) = \mathsf{sbf}^{\mathsf{rem}}_{R,\tau}(t) \stackrel{\mathsf{def}}{=} \max_{0 \le t' \le t} \big\{ \mathsf{sbf}_R(t') - \mathsf{rbf}^{\mathsf{ISR}}_{\tau}(t') \big\}.$$

Proof: Since the resource provided by R is first given to the release ISRs, the amount of resources available for executing τ is the remaining amount of resources after processing all the release ISRs. From Lemma 4, the amount of resources requested by the release ISRs over any interval of length t is at most $\mathsf{rbf}^{\mathsf{ISR}}_{\tau}(t)$. Therefore, after processing the ISRs over any interval of length t, there are at least $\mathsf{sbf}_R(t) - \mathsf{rbf}^{\mathsf{ISR}}_{\tau}(t)$ resources remaining. In addition, the amount of remaining resources over any interval of length t is always larger than, or equal to, the amount of remaining resources over an interval of length t', for all $0 \le t' \le t$. Hence, $\mathsf{sbf}^{\mathsf{rem}}_{R,\tau}(t)$ is the minimum remaining resource after processing the release ISRs over an interval of length t, which is also the amount of resources available to the tasks τ over an interval of length t. Thus, the tasks in τ are scheduled in presence of intra-component release interference overheads if they are schedulable by a resource R' with a supply bound function equal to $\mathsf{sbf}^{\mathsf{rem}}_{R,\tau}(t)$ when assuming no overheads. ∎

The next corollary follows directly from Lemma 5 and the analysis under zero overheads in Lemmas 1 and 2.

Corollary 6. *The release interference-aware schedulability conditions for a component with workload τ using a resource R for EDF and FP are given as follows.*

- **FP:** $\forall\, 1 \le i \le n\ \exists t \in [0, d_i]\ s.t.\ \mathsf{sbf}^{\mathsf{rem}}_{R,\tau}(t) \ge \mathsf{rbf}_{C,i}(t)$.
- **EDF:** $\forall\ 0 < t \le \mathsf{LCM}_\tau$, $\mathsf{sbf}^{\mathsf{rem}}_{R,\tau}(t) \ge \mathsf{dbf}_C(t)$, *where* LCM_τ *is the least common multiple of p_i for all $\tau_i \in \tau$.*

The overhead-aware schedulability analysis can now be derived based on the inflatable overhead-aware analysis and the release interference-aware test, which is given by Theorem 7. This theorem is a direct result of Lemmas 3 and 5.

Theorem 7. ***(Overhead-aware Schedulability Test)*** *A component $C = \langle \tau, A \rangle$ is schedulable by a resource R in the presence of platform overheads if the inflated workload τ' is schedulable under A by a resource R' in the absence of overheads, where the SBF of R' is equal to $\mathsf{sbf}^{\mathsf{rem}}_{R,\tau}(t)$.*

VI. Overhead-aware Interfaces and Interface Computation

This section describes our proposed compositional analysis method, which is based on the overhead-aware schedulability tests from the previous section. We begin by discussing the ISR amortization problem in deadline-sensitive interface models, such as EDP and PRM, which necessitates a new representation and computation method for the interface.

A. Challenge: ISR amortization problem

Based on the component overhead-aware schedulability test in Section V, it may seem possible to use an existing resource model, such as the EDP or the PRM model, to encapsulate the total resource requirements of both the tasks and their overheads. Alternatively, one could try to capture the release interference overhead (e.g., the release ISRs in Fig. 3) with a separate EDP or PRM model. However, using such resource models to capture the release interference overhead is unsafe, as illustrated by the following example.

ISR amortization under deadline-sensitive interfaces. Consider a component C with a workload $\tau = \big\{\tau_1 = (5, 4, 5); \tau_2 = \cdots = \tau_{51} = (500, 1, 500)\big\}$, which is scheduled under EDF. Let $\Delta^{rel} = 0.02$. (All times are in ms.) In this example, we consider only the release ISRs and assume that there are no other types of overheads.

By Corollary 6, the above component is unschedulable by a fully available processor (i.e., $\mathsf{sbf}_R(t) = t$) in presence of release interference overheads. (Note that the worst-case response time of τ_1 is 5.020, since it incurs a maximum release interference overhead of $51 \times 0.020 = 1.02$ ms.) However, if we abstract each of the tasks' resource demands and the intra-component release interference overheads into an EDP, we obtain $\Gamma_\tau = (5, 4.5, 4.5)$ and $\Gamma_{\mathsf{ISR}} = (1, 0.006, 1)$. Since these two interfaces are schedulable under FP, the system is deemed schedulable, which is incorrect. The same situation happens if we use other deadline-sensitive resource interfaces, such as PRM or SBF.

The flaw in the above analysis based on EDP abstraction comes from the ISR amortization, which is caused by the

interface deadline. When all release ISRs are invoked at $t = 0$, they are executed immediately and will keep the processor busy until $t = 0.02 * 51 = 1.02$. However, when the resources requested by the release ISRs (captured by $\mathsf{rbf}_\tau^{ISR}(t)$ in Lemma 4) are encapsulated in an EDP model, the request is amortized to 0.006 time units that need to be completed within every time unit. This effectively enables the processor to execute the jobs in τ that have been effectively released. Since tasks do not experience the full release interference overhead, they still make their deadlines in this example. In other words, when the release ISRs are included in an EDP resource model, the implicit assumption is that the release ISRs can be scheduled/delayed, which is impossible in practice.

Note that for task execution, deadline-sensitive resource models are a natural way to capture the tasks' resource demands, since a task $\tau_i = (p_i, e_i, d_i)$ meets its deadline as long as every job of τ_i can be given e_i time units within d_i time units from the instant it is released. Although the release ISRs of τ_i are also invoked every p_i time units and require up to Δ^{rel} time units each, they cannot be delayed by a task's execution, and their deadlines are unavailable (and ambiguous). In other words, the deadline-sensitive resource model is not a good fit for release ISRs – regardless of how the resource deadline or the period are chosen.

B. Overhead-aware interface representation

Our approach is to capture the overhead-aware resource requirements of the component using a dual-interface representation. Given a component $C = \langle \tau, A \rangle$ with a workload τ and scheduling algorithm A, an overhead-aware interface of C is given by $\langle I_\tau, I_{\mathsf{ISR}} \rangle$, where:

- I_τ is a resource model that captures the resource requirements of the tasks in τ, taking into consideration inflatable overheads;
- I_{ISR} is a resource model that captures the (non-inflatable) intra-component release interference overheads.

Since the inflatable overheads can be accounted for by the inflated WCETs of the tasks in τ, which can be scheduled, I_τ can be any deadline-sensitive resource interface (e.g., PRM, EDP, SBF). For ease of presentation, we use an EDP resource model to represent I_τ. On the other hand, due to the ISR amortization problem discussed above, we cannot represent I_{ISR} using a deadline-sensitive resource model. Instead, we represent it as a request bound function (RBF), which corresponds naturally to the resource requests of the ISRs. Thus, an interface of C is given by $\langle I_\tau, I_{\mathsf{ISR}} \rangle$ where I_τ is an EDP model and I_{ISR} is an RBF. We refer to this representation as the $\langle$EDP, RBF$\rangle$ interface model.

C. Generating interfaces for elementary components

Given an elementary component $C = \langle \tau, A \rangle$, with workload $\tau = \{\tau_1, \ldots, \tau_n\}$ and scheduling algorithm A (which can be EDF, RM or DM), where each $\tau_i = (p_i, e_i, d_i)$ is an explicit deadline periodic task, the overhead-aware interface of C can be computed based on the overhead-aware schedulability test in Theorem 7, using the following procedure:

- **Step 1.** Compute the inflated WCET e'_i of each task $\tau_i \in \tau$ to account for the inflatable overheads, using Eq. (7).
- **Step 2.** Generate a bandwidth-optimal EDP interface $\Gamma_{\tau'} = (\Pi, \Theta, \Delta)$ that can feasibly schedule the inflated workload $\tau' = \{(p_1, e'_1, d_1), \ldots, (p_n, e'_n, d_n)\}$ under the scheduling algorithm A, using the algorithms presented in [15].
- **Step 3.** Compute the request bound function $\mathsf{rbf}_\tau^{\mathsf{ISR}}(t)$ that bounds the intra-component release interference overheads of the workload τ, using Lemma 4.
- **Step 4.** The overhead-aware interface of C is given by $I_C = \langle I_\tau, I_{\mathsf{ISR}} \rangle$, where $I_\tau = \Gamma_{\tau'}$ and $I_{\mathsf{ISR}}(t) = \mathsf{rbf}_\tau^{\mathsf{ISR}}(t)$.

The next lemma states the feasibility of the interfaces computed with the above procedure. It is derived directly from the correctness of the overhead-aware schedulability test (Theorem 7) and the feasibility of the EDP interface in the zero-overhead setting [15].

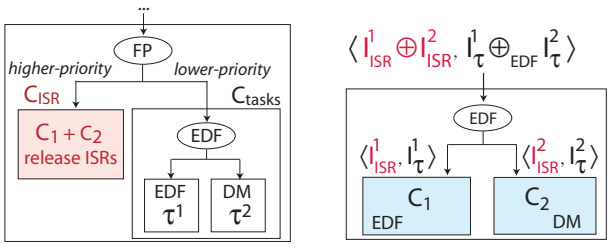
Lemma 8. *The overhead-aware interface $I_C = \langle I_\tau, I_{\mathsf{ISR}} \rangle$ obtained in Step 4 is a feasible interface for C – i.e., if I_τ and I_{ISR} are schedulable by a resource R under the schedulability test in Theorem 7, then the component C is schedulable under the same resource R in the presence of platform overheads.*

Example 1. *Consider $C = \langle \tau, EDF \rangle$, with $\tau = \{(10, 2, 10); (10, 1, 10); (20, 1, 20); (20, 5, 20)\}$. We assume that the WCETs of the tasks in τ have been inflated, and that a release ISR takes up to 0.02 time units. The overhead-aware interface of C is given by $I_C = \langle (\Pi = 10, \Theta = 6, \Delta = 6), \mathsf{rbf}_C \rangle$ where*

$$\mathsf{rbf}_C(t) = 0.02\left(\lceil \frac{t}{10} \rceil + \lceil \frac{t}{10} \rceil + \lceil \frac{t}{20} \rceil + \lceil \frac{t}{20} \rceil\right) = 0.04\left(\lceil \frac{t}{10} \rceil + \lceil \frac{t}{20} \rceil\right).$$

D. Interface composition

The interface of a composite component can be computed by pointwise composing the interfaces of their subcomponents. To establish this, we first consider the execution of a composite component, illustrated Fig. 4(a). In this example, C is composed of C_1 and C_2, which are scheduled under EDF, where $C_1 = \langle \tau^1, EDF \rangle$ and $C_2 = \langle \tau^2, DM \rangle$. In the presence of release interference overheads, the resources available to C are always first allocated to the release ISRs of the tasks in both C_1 and C_2, and the remaining resources are available for executing the tasks τ_1 and τ_2. This is illustrated in the figure by a higher-priority release ISRs component C_{ISR} and the lower-priority task component C_{tasks}.



(a) Execution on the platform: $C_1 = \langle \tau^1, \text{EDF} \rangle$; $C_2 = \langle \tau^2, \text{DM} \rangle$

(b) Interface composition

Fig. 4: Overhead-aware interface composition.

The computation of C's interface is illustrated by Fig. 4(b). In the figure, the interfaces of C_1 and C_2 are given by $\langle I^1_{\mathsf{ISR}}, I^1_\tau \rangle$ and $\langle I^2_{\mathsf{ISR}}, I^2_\tau \rangle$, respectively. Since the tasks of C_1 and C_2 are scheduled within the lower-priority component C_{tasks} as in the ideal case when there is no overhead (see Fig. 4(a)), we can compute their resource requirements by composing I^1_τ and I^2_τ under EDF scheduling. The release ISRs of both C_1 and C_2 are processed within the higher-priority component C_{release} as soon as they arrive. Therefore, their resource requests can be computed by composing the release ISR interfaces, I^1_{ISR} and I^2_{ISR}. The next theorem generalizes the above observation:

Theorem 9. *Let $C = \langle W = \{C_1, ..., C_n\}, A \rangle$, where the interface of C_i is $\langle I^i_\tau, I^i_{\mathsf{ISR}} \rangle$, for all $1 \leq i \leq n$. Then $I_C = \langle I_\tau, I_{\mathsf{ISR}} \rangle$ is a feasible overhead-aware interface of C, where*

- *I_τ is a (feasible) composition of I^1_τ, $I^2_\tau, \ldots, I^n_\tau$ under the scheduling algorithm A, assuming there are no platform overheads; and*
- *$I_{\mathsf{ISR}} = \sum_{i=1}^{n} I^i_{\mathsf{ISR}}$.*

When I^i_τ is an EDP resource model, the composition can be computed efficiently under EDF and DM scheduling, using the technique presented in [15].

Proof: The correctness of the theorem is based on two observations. First, by the definition of I_τ, if the interface I_τ is schedulable by a resource R when there is no overhead, then all the interfaces I^i_τ are also schedulable by R when there is no overhead. Since each I^i_τ includes the inflatable overheads, if I_τ is schedulable by R when there is no overhead, the tasks in C_i are schedulable in the presence of inflatable overheads if we assume zero release interference overhead.

Second, by definition, I^i_{ISR} is the RBF of the release ISRs of C_i. Since all the release ISRs are scheduled by the system as soon as they arrive (see the component C_{ISR} in Fig. 4), the amount of resources requested by all the release ISRs of the subcomponents, C_1 to C_n, over any interval of length t, is bounded by $\sum_{i=1}^{n} I^i_{\mathsf{ISR}}(t)$. In other words, I_{ISR} is the RBF of the release interference overheads of C.

If we combine the above observations, we can conclude that all C is schedulable by a resource R considering platform overheads if I_τ and I_{ISR} are schedulable by R according to the overhead-aware schedulability test in Theorem 7. ■

VII. Evaluation

To evaluate the effectiveness of the proposed overhead-aware compositional analysis, we performed simulations using randomly generated workloads. We had three main objectives for our evaluation: (1) Validate the accuracy of the overhead-aware analysis and evaluate its relative performance on a realistic platform (against a baseline approach and an existing analysis that ignores overheads); (2) study the effect of task parameters on the platform overheads; and (3) evaluate the performance of our method in terms of resource bandwidth savings against a baseline approach.

Baseline approach: The approach we used as a baseline for our evaluation accounts for overheads by inflating each task's WCET by all types of overheads, including the release interference overheads. Since a release ISR is a timer interrupt, we followed the interrupt accounting technique from [11]. The release interference overhead each task experiences was computed as

$$e^{rel}_i = \sum_{\tau_j \in \tau} \lceil \frac{p_i}{p_j} \rceil \Delta^{rel},$$

where Δ^{rel} is the maximum value of one release ISR (c.f. Section IV). The inflated WCET e''_i of each task was computed as $e''_i = e'_i + e^{rel}_i$, where e'_i is the WCET that already includes other types of overheads; this is computed using Eq. (7). We then applied the existing compositional analysis for EDP interfaces [15] on the inflated WCETs. Note that this baseline approach is safe, but it requires that every component be aware of the details of all tasks in the system.

A. Experimental setup

Workload. Our evaluation was performed on a set of synthetic real-time workloads. Each workload contained a set of randomly generated periodic task sets, with task periods uniformly distributed between 110ms and 1100ms (which is the same as the experiment in [20]). The tasks' deadlines are equal to their periods. The tasks' utilizations follow a uniform distribution within the range [0.02%,0.5%] and three bimodal distributions, where the utilizations were distributed uniformly over either [0.02%,0.5%] or [0.5%,10%], with respective probabilities of $8/9$ and $1/9$ (light), $6/9$ and $3/9$ (medium), and $4/9$ and $5/9$ (heavy)[2]. Since each overhead value is typically much smaller than a task's WCET, small task utilization values were used to obtain more tasks per task set (so as to better observe the effects of the overheads). Each generated workload was then distributed uniformly into a set of components of a two-level scheduling hierarchy. Each component's scheduling algorithm was chosen to be either EDF or DM.

Overhead value measurement. For the theoretical analysis, we used the maximum observed value for each type of overhead, based on measurements on our experimental platform using the feather trace tool from LITMUS [10]. To be conservative, we measured the overheads using large task sets with 1000 tasks each. The obtained values were $\Delta^{rel} = 13.727\mu s$, $\Delta^{sch} = 36.565\mu s$, $\Delta^{cxs} = 86.917\mu s$, $p_{\mathsf{tick}} = 1ms$, $\Delta^{\mathsf{tick}} = 4.727\mu s$, and $\Delta^{\mathsf{CRPD}_i} = 139.12\mu s$ for all tasks τ_i.

Experimental platform. Our platform evaluation was performed on the RT-Xen 0.3 platform [32] on a Dell Optiplex-980 quad-core processor, using the same experimental setup as in [20]. Since our analysis does not consider the components' release and preemption overheads, we pinned the *Domain 0* to Core 0 and all the guest domains to Core 1 to avoid these overheads in our experiment. Each component in the scheduling hierarchy was then mapped to a guest domain. Hence, all tasks ran on the same processor, which is consistent with our analysis setting.

[2]The distribution probabilities are similar to the ones used in [9]

Evaluation method. For each generated workload, we computed the component interfaces using three different approaches: 1) our overhead-aware interface computation from Section VI, 2) the baseline accounting method that was described above, and 3) the conventional EDP interface computation [15], which ignores overheads. We then analyzed the components' schedulability and computed the resource bandwidth required to feasibly schedule each component in the hierarchy for each of the three methods. For the baseline accounting and the conventional EDP analysis methods, which use EDP interfaces, the resource bandwidth required by an interface is the same as the interface's bandwidth. For our overhead-aware analysis method, which uses ⟨EDP, RBF⟩ interfaces, we computed the minimum-bandwidth EDP resource model that can feasibly schedule the interface based on the schedulability test in Theorem 7 and the supply bound function of an EDP model (c.f. Section II).

We also simulated the same workloads on RT-Xen [20] and measured the deadline miss ratio for each of the three methods. Each domain corresponds to a component in a workload, and its budget is assigned to be the theoretically calculated resource bandwidth required by the component's interface.

B. Results

Performance in theory vs. in practice. For this experiment, we generated a set of workloads with workload utilizations ranging from 0.1 to 1.1, with a step of 0.1. Each workload contained 25 independently generated task sets; the task utilizations were uniformly distributed, as described earlier. We then computed the fraction of schedulable task sets for each workload in theory (according to the schedulability test) and in practice (according to the measurements on the RT-Xen platform).

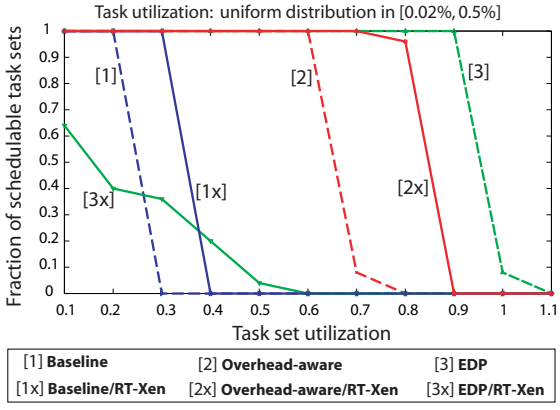


Fig. 5: Fraction of schedulable task sets vs. workload utilization.

Fig. 5 shows the fraction of schedulable task sets with respect to workload utilization for the three analysis approaches. The dotted lines represent the theoretical values and the solid lines represent the measured values. We observe the following from the evaluation results:

Correctness of our overhead-aware analysis: The experimental results confirm that our overhead analysis can correctly estimate the resource requirements of the system in practice. As is shown in Fig. 5, the fraction of schedulable task sets computed theoretically using our overhead-aware analysis always safely upper-bounds the actual schedulable fraction observed on the platform. We note that, since our analysis estimates the resource needs based on the worst-case scenario, there are some task sets that are deemed unschedulable but may still be schedulable in practice.

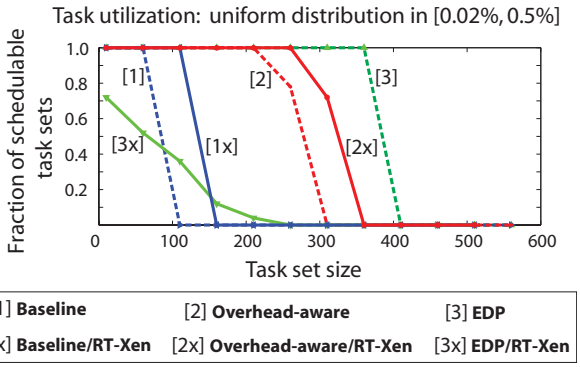


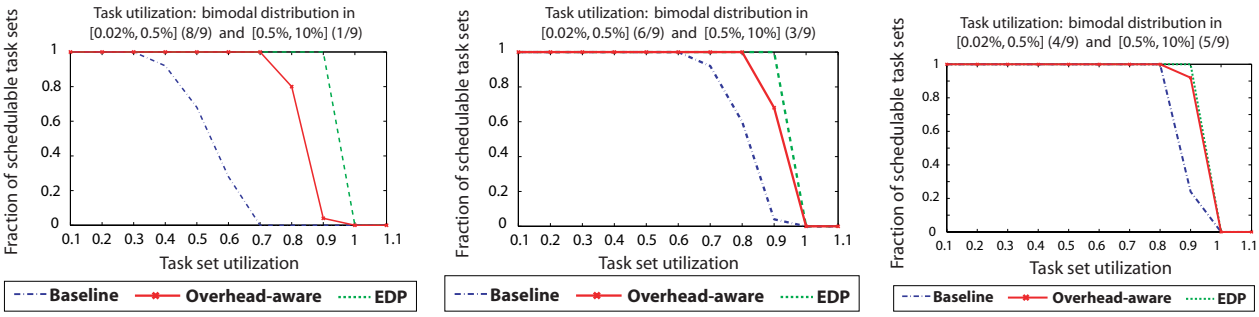
Fig. 6: Fraction of schedulable task sets vs. task-set size.

Invalidity of EDP analysis results: We observe that the EDP analysis method (which ignores platform overheads) significantly underestimates the components' resource requirements and leads to many task sets missing their deadlines when they are run on the platform. This is visible in Fig. 5, where the theoretical values for the schedulable fraction fall strictly above the measured ones. For example, although all task sets are predicted to be schedulable for the workload with 0.9 utilization, all of them are in fact unschedulable when scheduled on the platform under the computed interfaces.

Pessimism of the baseline approach: Although our results confirm that the baseline approach is safe, they also show that it is overly pessimistic. Since the baseline approach overestimates the resource overheads, it predicts that only a small fraction of task sets is schedulable, although more task sets can in fact be scheduled (see Fig. 5). We also note the effect of this pessimism in practice: since the baseline approach overestimates the resource requirements of components, the resource budgets given to higher-priority components are larger than necessary; therefore, the lower-priority components may not get sufficient budgets, and their tasks are more likely to miss their deadlines. As a result, the number of task sets that are schedulable decreases, as is confirmed by the measured fraction of the baseline approach in Fig. 5.

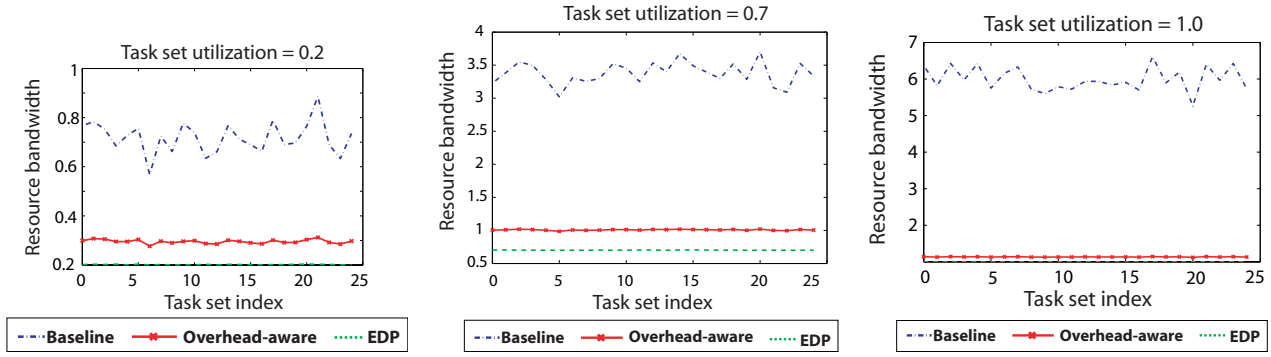
In short, our overhead-aware analysis performs best compared to the other two approaches in terms of accuracy and resource utilization in both theory and practice.

Impact of the number of tasks on overheads. Since the release interference overheads can accumulate as the number of tasks in the system increases, we quantified the impact of the number of tasks on schedulability as follows: we varied the number of tasks per task set, here called the *task-set size*, from 10 to 510, with an increment step of 50; for each task-set size, we generated a workload that consisted of 50 task sets, using the uniform task utilization distribution. We then derived the fraction of schedulable task sets in the workload, computed theoretically and measured on RT-Xen platform, for each of the three analysis approaches.



(a) Bimodal-light distribution (b) Bimodal-medium distribution (c) Bimodal-heavy distribution

Fig. 7: Fraction of schedulable task sets with different task utilizations.



(a) Low load: task set utilization = 0.2 (b) High load: task set utilization = 0.7 (c) Overload: task set utilization = 1.0

Fig. 8: Resource bandwidth requirements under different load situations.

As is shown in Fig. 6, the fraction of schedulable tasks decreases for all approaches as the number of tasks in the system increases. However, our approach again outperforms the baseline approach: with our approach, the fraction of schedulable task sets remains at 100% until the number of tasks reaches 210, whereas the fraction of the baseline approach falls to 0% when the number of tasks reaches 160. The main reason for this is that our method provides a more accurate estimate of the resource overheads, and can thus achieve a much tighter schedulability test.

Impact of task utilization. We also evaluated the effect of task utilization distribution on the platform overheads. Fig. 7 shows the fraction of schedulable task sets under three different bimodal distributions (c.f. Section VII-A). We observed that, as the task's utilization distribution was changed from bimodal-light to bimodal-medium to bimodal-heavy, the difference in the schedulable ratios of the three analysis methods became smaller. This is because the number of tasks in the system decreases as the tasks' utilization increases, and this results in less platform overhead.

Resource bandwidth savings. Fig. 8(a–c) shows the resource bandwidth that each task set requires with each of the three approaches, for three different utilization values: U = 0.2 (low load), U = 0.7 (high load) and U = 1.0 (overload). The graphs show that our overhead-aware analysis consistently outperforms the baseline approach in terms of bandwidth savings. For example, the interfaces computed using the baseline approach required on average 2.4, 3.3, and 5 times the resource bandwidth our interfaces required in the low-load, high-load and overload scenarios, respectively. Similarly, in the high-load scenario, all the task sets were deemed unschedulable by the baseline analysis (their interface bandwidths exceeded 1), even though they were schedulable. At the same time, the resource bandwidths required by the different task sets with the same utilization under our accounting method were approximately the same, in contrast to those of the baseline approach. This illustrates that our accounting method is less dependent on the tasks' timing parameters than the baseline approach. Note that, in the absence of overhead, the interfaces always require less resource bandwidth, but, as shown earlier, they are not safe in practice.

VIII. Related Work

Several compositional schedulability analysis techniques have been developed (see e.g., [4], [7], [15], [24], [28]); however, they focus primarily on improving the theoretical analysis tradeoffs between interface representation and interface computation. Although theoretical overhead, i.e., the resource overhead caused by the interface abstraction, is a well-known problem and by now is relatively well-understood [13], [16], [20], platform overheads are typically ignored in existing CSA theories.

The gap between theory and practice of CSA has also been recognized and addressed via platform supports for compositional scheduling, such as on VxWorks [6], μC/OS-II [30], and virtualization [20], [33] platforms. Hierarchical scheduling for closed systems has also been implemented on several OS kernels (e.g. [14], [26], [31]). These implementations offer some solutions to platform-related issues, such as server design and quantization. Platform overheads have also been studied

through measurements [2], [33], but they have not yet been incorporated into the interface model and interface computation. Although Behnam et al. [6] considers release overheads in the response time analysis, the method in [6] cannot be applied to compositional systems, since it assumes that all the task information is given a priori. In addition, the work in [23] accounts for components' context switching overheads in the selection of the components' interfaces; however, it does not consider overheads that a job experiences during its lifetime, which is a focus of our work.

A closely related line of work is developing overhead accounting techniques for closed systems. Most notably, Brandenburg [9] has proposed an elegant method for accounting various sources of platform overheads. Our work extends the original method from [11] for use with open systems and to enable incremental interface analysis. Our work also leverages existing work on cache effects analysis [1], [5] for the computation of the cache-related preemption delay (CRPD). Note that, unlike ours, these works focus on cache effects and on closed systems. Also in the closed setting, Harbour and Palencia [19] models release interferences as tasks' release jitters, and it uses the maximum jitter values for the worst-case response time analysis. The method in [19] cannot be applied to a compositional setting, however, since it is not possible to obtain a safe bound on the jitter of a task that is caused by release ISRs of other components (which are not known a priori).

IX. Conclusion

We have presented an overhead-aware compositional analysis technique for real-time systems. Our technique accounts for platform overheads in the component interfaces, and thus enables a safe application of compositional scheduling theories in practice. We have described a method for accounting different types of platform overheads in an open system, as well as a new overhead-aware schedulability test for components. We have also introduced an interface model that captures the overhead information in a succinct manner, and its accompanying interface computation methods. Our evaluation on synthetic workloads shows that, unlike existing analysis theories, our theoretical interfaces are safe in practice, and compared to a baseline approach based on WCET inflation, our analysis can help reduce resource bandwidth by up to a factor of five.

Considering platform overheads in compositional analysis theory is crucial to guaranteeing correct timing behaviors of compositional scheduling systems in practice. Since the proposed overhead-aware interfaces cleanly separate inflatable from non-inflatable overheads, it seems feasible to extend them to the multicore setting. An interesting area for future work is to leverage multicore resource models, such as the multiprocessor periodic resource model [16], for the inflatable overheads while extending the release interference overhead accounting method to non-inflatable overheads on a multicore platform. We also plan to perform an extensive evaluation of the overhead-aware interfaces on a range of processor platforms and to conduct case studies in the automotive and avionic domains (e.g., ARINC software [3]).

References

[1] S. Altmeyer, R. I. Davis, and C. Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, Sep. 2012.

[2] M. Åsberg, T. Nolte, S. Kato, and R. Rajkumar. ExSched: An external cpu scheduler framework for real-time systems. In *RTCSA*, 2012.

[3] Avionics Electronic Engineering Commitee. Avionics Application Software Standard Interface: Part 1 - Required Services (ARINC Specification 653P1-3), Nov. 2010.

[4] S. Baruah and N. Fisher. Component-based design in multiprocessor real-time systems. In *ICESS*, 2009.

[5] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *OSPERT*, 2010.

[6] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. J. Bril. Towards hierarchical scheduling on top of VxWorks. In *OSPERT*, 2008.

[7] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *EMSOFT*, 2007.

[8] D. Bovet and M. Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 3 edition, 2005.

[9] B. B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

[10] B. B. Brandenburg and J. H. Anderson. Feather-trace: A light-weight event tracing toolkit. In *OSPERT*, 2007.

[11] B. B. Brandenburg, H. Leontyev, and J. H. Anderson. An overview of interrupt accounting techniques for multiprocessor real-time systems. *Journal of Systems Architecture*, 57(6):638–654, Jun. 2011.

[12] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson. LITMUS RT: A testbed for empirically comparing real-time multiprocessor schedulers. In *RTSS*, 2006.

[13] S. Chen, L. T. X. Phan, J. Lee, I. Lee, and O. Sokolsky. Removing abstraction overhead in the composition of hierarchical real-time systems. In *RTAS*, 2011.

[14] M. Danish, Y. Li, and R. West. Virtual-CPU scheduling in the Quest operating system. In *RTAS*, 2011.

[15] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *RTSS*, 2007.

[16] A. Easwaran, I. Shin, and I. Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25–59, Sep. 2009.

[17] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *RTSS*, 2002.

[18] N. Fisher and F. Dewan. Approximate bandwidth allocation for compositional real-time systems. In *ECRTS*, 2009.

[19] M. G. Harbour and J. C. Palencia. Response time analysis for tasks scheduled under edf within fixed priorities. In *RTSS*, 2003.

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

[21] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS*, 1989.

[22] H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Real-Time Systems*, 43(1):60–92, Sep. 2009.

[23] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *ECRTS*, 2003.

[24] G. Lipari and E. Bini. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In *RTSS*, 2010.

[25] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[26] G. Parmer and R. West. HiRes: A system for predictable hierarchical resource management. In *RTAS*, 2011.

[27] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *ECRTS*, 2010.

[28] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, 2003.

[29] D. L. Sun, Z. Deng, J. W. -S, and L. J. Sun. Dynamic scheduling of hard real-time applications in open system environment. 1996.

[30] M. M. H. P. van den Heuvel, R. J. Bril, J. J. Lukkien, and M. Behnam. Extending a HSF-enabled open-source real-time operating system with resource sharing. In *OSPERT*, 2010.

[31] Y. Wang and K. Lin. The implementation of hierarchical schedulers in the RED-Linux scheduling framework. In *ECRTS*, 2000.

[32] S. Xi, J. Lee, C. Lu, C. D. Gill, S. Chen, L. T. X. Phan, O. Sokolsky, and I. Lee. RT-Xen: Real-time virtualization based on hierarchical scheduling. http://sites.google.com/site/realtimexen/.

[33] J. Yang, H. Kim, S. Park, C. Hong, and I. Shin. Implementation of compositional scheduling framework on virtualization. In *SIGBED Rev.*, 2011.