A Compositional Scheduling Framework for Digital Avionics Systems

Arvind Easwaran
CISTER/IPP-HURRAY
Polytechnic Institute of Porto
Portugal
aen@isep.ipp.pt

Insup Lee, Oleg Sokolsky
Department of CIS
University of Pennsylvania
PA, 19104, USA
{lee,sokolsky}@cis.upenn.edu

Steve Vestal
Boston Scientific
MN 55112, USA
stephen.vestal@bsci.com

***Abstract*—ARINC specification 653-2 describes the interface between application software and underlying middleware in a distributed real-time avionics system. The real-time workload in this system comprises of partitions, where each partition consists of one or more processes. Processes incur blocking and preemption overheads, and can communicate with other processes in the system. In this work, we develop compositional techniques for automated scheduling of such partitions and processes. At present, system designers manually schedule partitions based on interactions they have with the partition vendors. This approach is not only time consuming, but can also result in under utilization of resources.**

I. INTRODUCTION

ARINC standards, developed and adopted by the *Engineering Standards for Avionics and Cabin Systems* committee, deliver substantial benefits to airlines and aviation industry by promoting competition, providing inter changeability, and reducing life-cycle costs for avionics and cabin systems. In particular, the 600 series ARINC specifications and reports define enabling technologies that provide a design foundation for digital avionics systems. Within the 600 series, this work deals with ARINC specification 653-2, part I [3] (henceforth referred to as ARINC-653), which defines a general-purpose Application/Executive (APEX) software interface between the operating system of an avionics computer and the application software.

As described in ARINC-653, the real-time system of an aircraft comprises of one or more *core modules* connected with one another using switched Ethernet. Each core module is a hardware platform that consists of one or more processors among other things. They provide space and temporal partitioning for independent execution of avionics applications. Each independent application is called a *partition*, and each partition in turn is comprised of one or more *processes* representing its real-time resource demand. The workload on a single processor in a core module can therefore be described as a two-level hierarchical real-time system. Each partition comprises of one or more processes that are scheduled among themselves using a (local) partition specific scheduler. All the partitions that are allocated to the same processor are then scheduled among themselves using a (global) partition level scheduler. For example, Figure 1 shows two such systems, where partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ are scheduled together under a global scheduler on one processor, and partitions $\mathcal{P}_3$ and $\mathcal{P}_4$ are scheduled together under a global scheduler on another processor. Each partition $\mathcal{P}_i$ in turn is comprised of processes $\tau_{i,1}, \ldots, \tau_{i,m_i}$, scheduled under a local scheduler[1]. Processes are periodic tasks that communicate with each other. Sequences of such communicating processes form dependency chains, and designers can specify end-to-end latency bounds for them. For example, Figure 1 shows one such chain between tasks $\tau_{1,1}, \tau_{2,2}$, and $\tau_{3,2}$. Processes within a partition can block each other using *semaphores* for access to shared data, giving rise to blocking overhead (tasks $\tau_{4,2}$ and τ_{4,m_4} in the figure). Further, processes and partitions can also be preempted by higher priority processes and partitions, respectively, resulting in preemption overheads.

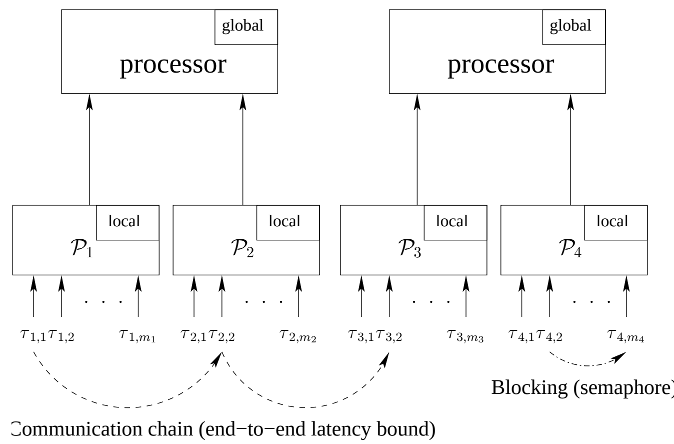


Fig. 1. Scheduling hierarchy for partitions

There are several problems related to the hierarchical system described above that must be addressed. For scheduling partitions, it is desirable to abstract the communication dependencies between processes using parameters like offsets, jitter, and constrained deadlines. This simplifies a global processor and network scheduling problem into several local single processor scheduling problems. The process deadlines must also guarantee satisfaction of end-to-end latency bounds

[1]The local scheduler can be different from the global scheduler and each of the other local schedulers.

specified by the designer. Given such processes we must then generate scheduling parameters for partitions, to be used by the global scheduler. The resulting global schedule must provide sufficient processor capacity to schedule processes within partitions. Furthermore, these scheduling parameters must also account for blocking and preemption overheads incurred by processes and partitions.

This avionics system frequently interacts with the physical world, and hence is subject to stringent government regulations. Then, to help with system certification, it is desirable to develop schedulability analysis techniques for such hierarchical systems. Furthermore, these analysis techniques must account for resource overheads arising from preemptions, blocking, and communication. In order to protect the intellectual property rights of partition vendors, it is also desirable to support partition isolation; only so much information about partitions must be exposed as is required for global scheduling and the corresponding analysis. We therefore consider *compositional* techniques for partition scheduling, i.e., we schedule partitions and check their schedulability by composing interfaces, which abstractly represent the resource demand of processes within partitions.

Partition workloads can be abstracted into interfaces using existing compositional techniques [18], [12], [24], [9]. These techniques use resource models as interfaces, which are models characterizing resource supply from higher level schedulers. In the context of ARINC-653, these resource model based interfaces can be viewed as abstract resource supplies from the global scheduler to each partition. Various resource models like periodic [18], [24], bounded-delay [12] and Explicit Deadline Periodic (EDP) [9], have been proposed in the past. However, before we can use these techniques, we must modify them to handle ARINC-653 specific issues like communication dependencies, and blocking and preemption overheads. In this paper, we assume that communication dependencies and end-to-end latency bounds are abstracted using existing techniques into process parameters like offset, jitter, and constrained deadline (see [25], [22]). Note that although we do not present solutions to this problem, it is however important, because it motivates the inclusion of aforementioned process parameters.

Contributions. In this paper we model ARINC-653 as a two-level hierarchical system, and develop compositional analysis techniques for the same. This is a principled approach for scheduling ARINC-653 partitions that provides separation of concerns among different partition vendors, and therefore should facilitate system integration. In particular, our contributions can be summarized as follows:

1) We extend and improve existing periodic [18] and EDP [9] resource model based compositional analysis techniques to take into account (a) process communications modeled as offsets, jitter, and constrained deadlines, and (b) process preemption and blocking overheads. Section III presents this solution, and illustrates its effectiveness using actual workloads from avionics systems.
2) We develop techniques to schedule partitions using their interfaces, taking into account preemption overheads incurred by partitions. Specifically, in Section IV, we present a technique to count the exact number of preemptions incurred by partitions in the global schedule.

II. System model and related work

Partitions and processes. Each partition has an associated period that identifies the frequency with which it executes, *i.e.*, it represents the partition interface period. Typically, this period is derived from the periods of processes that form the partition. In this work, we assume that partitions are scheduled among themselves using deadline-monotonic (DM) scheduler [17]. This enables us to generate a static partition level schedule at design time (hyper-period schedule), as required by the specification. Processes within a partition are assumed to be periodic tasks[2]. ARINC-653 allows processes to be scheduled using preemptive, fixed priority schedulers, and hence we assume that each partition also uses DM to schedule processes in its workload.

As discussed in the introduction, we assume that communication dependencies and end-to-end latency requirements are modeled with process offsets, jitter, and constrained deadlines. Hence, each process can be specified as a constrained deadline periodic task $\tau = (\mathrm{O}, \mathrm{J}, \mathrm{T}, \mathrm{C}, \mathrm{D})$, where O is offset, J is jitter, T is period, C is worst case execution time, and $\mathrm{D}(\leq \mathrm{T})$ is deadline. Jobs of this process are *dispatched* at time instants $x\,\mathrm{T} + \mathrm{O}$ for every non-negative integer x, and each job will be *released* for execution at any time in the interval $[x\,\mathrm{T} + \mathrm{O}, x\,\mathrm{T} + \mathrm{O} + \mathrm{J}]$. For such a process it is reasonable to assume that $\mathrm{O} \leq \mathrm{D}$ [25]. Furthermore, we denote as $\langle\{\tau_1, \ldots, \tau_n\}, \mathrm{DM}\rangle$, a partition $\mathcal{P}$ comprising of processes $\tau_1, \ldots, \tau_n$ and using scheduler DM. Without loss of generality we assume that τ_i has higher priority than τ_j for all $i < j$ under DM.

In addition to the restrictions specified so far, we make the following assumptions for the system described herein. These assumptions have been verified to exist in avionics systems. (1) The processes within a partition, and hence the partition itself, cannot be distributed over multiple processors. (2) Periods of partitions that are scheduled on the same processor are *harmonic*[3]. This assumption has been verified to be true in digital avionics systems. For example, see the avionics workloads given in the appendix of this technical report [10]. Note that this assumption does not prevent processes from having non-harmonic periods. (3) Processes in a partition cannot block processes in another partition. This is because mutual exclusion based on semaphores require use of shared memory which can only happen within a partition.

Related work. Traditionally, the partition scheduling problem has been addressed in an ad-hoc fashion based on interactions between the system designer and vendors who

[2]Partitions with aperiodic processes also exist in avionics systems, but they are scheduled as background workload. Hence, we ignore them.

[3]A set of numbers $\{\mathrm{T}_1, \ldots, \mathrm{T}_n\}$ is harmonic if and only if, for all i and j, either T_i divides T_j or T_j divides T_i.

provide the partitions. Although many different ARINC-653 platforms exist (see [1], [2]), there is little work on automatic scheduling of partitions [15], [16], [21]. Kinnan *et. al.* [15] only provide preliminary heuristic guidance, and the other studies [16], [21] use constraint-based approaches to look at combined network and processor scheduling. In contrast to this high-complexity holistic analysis, we present an efficient compositional analysis technique that also protects intellectual property through partition isolation.

Resource models based on periodic resource allocations, and compositional analysis techniques using them, have been developed in the past [24], [9], [18]. However, these studies do not consider dependencies between and within partitions. But, such dependencies in hierarchical systems have been addressed in other studies [4], [7], [20], [8], [5], [13]. Almeida and Pedreiras [4] have presented compositional analysis techniques for the case when processes in partition workload have jitter in their releases. Davis and Burns [7] have extended this technique to consider release jitter as well as preemption overheads. Various resource-sharing protocols (HSRP [8], SIRAP [5], BROE [13]) that bound the maximum resource blocking time for dependent partitions have also been proposed in the past. However, all these approaches do not consider process offsets, which are used to model communication dependencies. Although these techniques can still be used for processes being considered in this paper, the analysis will be pessimistic in general. In this work, we address this issue by developing exact schedulability conditions for processes with offsets.

Matic and Henzinger [20] have also developed compositional analysis techniques in the presence of partition dependencies. They assume dependencies are modeled using one of the following two semantics: *Real-time workshop (RTW)*, and *Logical execution time*. Although RTW semantics is similar to the dependency constraints that we consider in our case study, it is more restrictive in that periods of dependent processes are required to be harmonic.

Mataix *et. al.* [6] compute the number of preemptions when partitions are scheduled under a fixed priority scheduler. However, unlike our technique which counts the preemptions exactly, they only present an upper bound.

III. Partition interface generation

In this section we propose techniques to compute a periodic or EDP resource model based interface for a partition $\mathcal{P} = \langle\{\tau_1,\dots,\tau_n\},\text{DM}\rangle$. We assume that $\Pi_{\mathcal{P}}$ denotes the interface period specified by system designer for $\mathcal{P}$. We first briefly discuss shortcomings of existing resource model based analysis, and then develop techniques that overcome these shortcomings.

A. Inadequacy of existing analysis

A periodic process such as the one described earlier, consists of an infinite set of real-time jobs that are required to meet temporal deadlines. The resource request bound function of a process upper bounds the amount of computational resource required to meet all its temporal deadlines ($\text{rbf} : \Re \rightarrow \Re$).

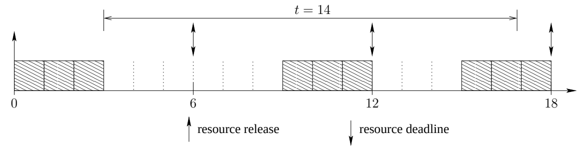


Fig. 2. Resource supply of model $\phi = \langle 6,3\rangle$ corresponding to $\text{sbf}_\phi(14)$

Similarly, the request bound function of a partition is the worst-case amount of resource requested by all the processes in the partition. We denote by $\text{rbf}_{\mathcal{P},i}(t)$, the request bound function of process τ_i in partition $\mathcal{P}$ for a time interval length t. Then, Equation (1) gives $\text{rbf}_{\mathcal{P},i}$ assuming that jitter and offset for all processes is zero [24].

$$\text{rbf}_{\mathcal{P},i}(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{\text{T}_j} \right\rceil \text{C}_j \qquad (1)$$

When processes have non-zero jitter but zero offset, Tindell and Clark have derived a critical arrival pattern which can be used to compute rbf [26]. In this arrival pattern each higher priority process is released simultaneously with the process under consideration, incurring maximum possible jitter. All future instances of these higher priority processes are released as soon as possible, *i.e.*, they incur zero jitter. Furthermore, the process under consideration itself is assumed to incur maximum possible jitter. Thus, for a process τ_i with zero offset but non-zero jitter, $\text{rbf}_{\mathcal{P},i}$ can be specified as

$$\text{rbf}_{\mathcal{P},i}(t) = \sum_{j=1}^{i} \left(\left\lceil \frac{t+\text{J}_j}{\text{T}_j} \right\rceil \text{C}_j \right) \qquad (2)$$

To satisfy the demand of a process or partition, the core module processor must supply sufficient computational resources. A resource model is a model for specifying the timing properties of this resource supply. For example, a resource supply that provides Θ units of resource every Π units of time can be represented using the periodic resource model $\phi = \langle\Pi,\Theta\rangle$ [24]. Similarly, a resource supply that provides Θ units of resource within Δ units of time, with this pattern repeating every Π time units can be represented using the explicit deadline periodic (EDP) resource model $\eta = \langle\Pi,\Theta,\Delta\rangle$ [9]. In both these models, $\frac{\Theta}{\Pi}$ represents resource bandwidth; average processor supply used over time. The supply bound function of a resource model lower bounds the amount of resource that the model supplies ($\text{sbf} : \Re \rightarrow \Re$). Given a resource model R and time interval length t, $\text{sbf}_R(t)$ gives the minimum amount of resource that R is guaranteed to supply in any time interval of length t. sbf for periodic (Equation (3)) and EDP (Equation (4)) resource models are reproduced below. In these equations $x_1 = 2(\Pi - \Theta)$, $y_1 = \left\lfloor \frac{t-(\Pi-\Theta)}{\Pi} \right\rfloor$, $x_2 = \Pi + \Delta - 2\,\Theta$, and $y_2 = \left\lfloor \frac{t-(\Delta-\Theta)}{\Pi} \right\rfloor$, where x_1 and x_2 are called *blackout intervals* for periodic and EDP models, respectively. These functions are also plotted in Figure 3. As shown in the figure, after the blackout interval, the models provide Θ units of resource in every Π time units. Corresponding to the sbf value for each interval length, there exists a resource supply pattern which generates that value. For example, the resource supply pattern of a periodic model $\phi = \langle 6,3\rangle$ corresponding to $\text{sbf}_\phi(14)$ is shown in Figure 2. Resource supply in the first period (interval $(0,6]$) is assumed to be given as soon as possible, and all successive supplies

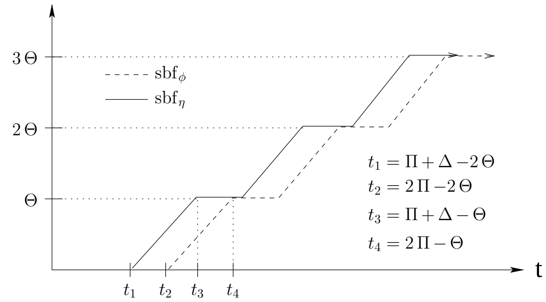


Fig. 3. sbf for periodic and EDP models

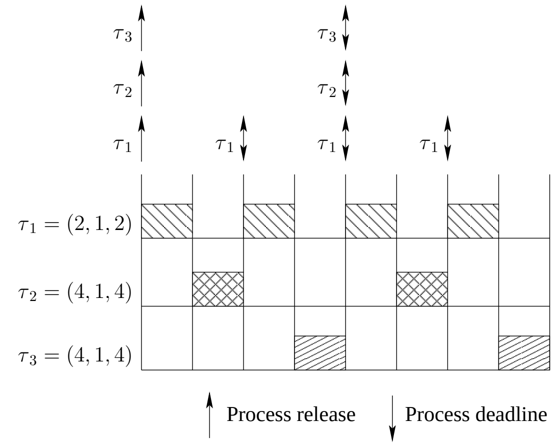


Fig. 4. Tasks with harmonic periods

are assumed to be given as late as possible.

$$\mathrm{sbf}_\phi(t) = \begin{cases} \max\{0, t - x_1 - y_1\,\Pi\} + y_1\,\Theta & t \ge \Pi - \Theta \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

$$\mathrm{sbf}_\eta(t) = \begin{cases} \max\{0, t - x_2 - y_2\,\Pi\} + y_2\,\Theta & t \ge \Delta - \Theta \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

When processes in a partition have zero offset and jitter values, conditions for schedulability of the partition using a periodic or EDP resource model have been proposed in the past [24], [9]. These conditions can be easily extended for processes with non-zero jitter, and is presented below.

Theorem 1: A partition $\mathcal{P} = \langle\langle \tau_1 = (0, \mathrm{J}_1, \mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (0, \mathrm{J}_n, \mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\rangle, \textsc{dm}\rangle$, where τ_j has higher priority than τ_k for all $j < k$, is schedulable over a periodic or EDP resource model R iff

$$\forall i, 1 \le i \le n, \exists t_i \in (0, \mathrm{D}_i - \mathrm{J}_i] \text{ s.t. } \mathrm{rbf}_{\mathcal{P},i}(t_i) \le \mathrm{sbf}_R(t_i),$$

where $\mathrm{rbf}_{\mathcal{P},i}$ is as defined in Equation (2).

Periodic or EDP resource model based interface for partition $\mathcal{P}$ can be generated using Theorem 1 as follows. We first set the period of resource model R to be equal to $\Pi_\mathcal{P}$. If R is a periodic resource model, then techniques presented in [24] can be used to develop a periodic model based interface. Since we are interested in minimizing processor usage (and hence resource bandwidth), we must compute the smallest Θ that satisfies this theorem. Hence, for each process τ_i, we solve for different values of t_i and choose the smallest Θ among them. Note that the theorem needs to be evaluated only at those time instants at which $\mathrm{rbf}_{\mathcal{P},i}$ changes. Θ for model R is then given by the largest value of Θ among all processes in $\mathcal{P}$. Similarly, if R is an EDP resource model then Easwaran *et. al.* [9] have presented a technique that uses this theorem to compute a resource model having smallest bandwidth. However, as described in the introduction, processes can be more accurately modeled using non-zero offset values. Then, a major drawback in using the aforementioned techniques is that Theorem 1 only gives sufficient schedulability conditions. This follows from the fact that the critical arrival pattern used by Equation (2) is pessimistic for processes with non-zero offset. Additionally, these techniques do not take into account preemption and blocking overheads incurred by processes.

In the following sections we extend Theorem 1 to accommodate processes with non-zero offsets, as well as to account for blocking and preemption overheads. Recollect from Section II that all the partitions scheduled on a processor are assumed to have harmonic interface periods. This observation leads to a tighter supply bound function for periodic resource models when compared to the general case. Therefore, we first present a new sbf for periodic resource models, and then extend Theorem 1.

B. sbf *under harmonic interface periods*

In the technique described in [24], a periodic interface $\phi = \langle \Pi, \Theta \rangle$ is transformed into a periodic task $\tau_\phi = (\Pi, \Theta, \Pi)$[4], before it is presented to the global scheduler. Note that the period of model ϕ and task τ_ϕ are identical, and period (Π) of task τ_ϕ is identical to its relative deadline. For the ARINC-653 partitions, this means that partitions scheduled on a processor are abstracted into periodic tasks with harmonic periods. When such implicit deadline[5] periodic tasks are scheduled under DM, every job of a task is scheduled in the same time instants within its execution window. This can be derived from the following observations: 1) whenever a job of a task is released, all the higher priority tasks also release a job at the same time, and 2) each job always executes for its stated worst-case execution time, in order to provide sufficient supply to the underlying periodic resource model. For example, Figure 4 shows the schedule for a periodic task set $\{\tau_1 = (2, 1, 2), \tau_2 = (4, 1, 4), \tau_3 = (4, 1, 4)\}$. It can be seen that every job of τ_3 is scheduled in an identical manner within its execution window.

Whenever task τ_ϕ is executing, the resource is available for use by periodic model ϕ. This means that resource supply allocations for ϕ also occur in an identical manner within intervals $(n\,\Pi, (n+1)\,\Pi]$, for all $n \ge 0$. In other words, the blackout interval x_1 in sbf_ϕ can never exceed $\Pi - \Theta$. For the example shown in Figure 4, assuming task τ_3 is transformed from a periodic resource model $\phi_3 = \langle 4, 1 \rangle$, the blackout interval for ϕ_3 can never exceed 3. Therefore, the general sbf for periodic models given in Equation (3) is pessimistic for

[4]This task is similar to the constrained deadline periodic task defined in Section II, except that it has zero jitter and offset.

[5]Tasks with $\mathrm{D} = \mathrm{T}$.

our case. Improved sbf_ϕ is defined as follows.

$$\mathrm{sbf}_\phi(t) = \left\lfloor \frac{t}{\Pi} \right\rfloor \Theta + \max\left\{0, t - (\Pi - \Theta) - \left\lfloor \frac{t}{\Pi} \right\rfloor \Pi \right\} \quad (5)$$

For a EDP resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$, the blackout interval in sbf_η is $\Pi + \Delta - 2\Theta$ [9]. Since $\Delta \geq \Theta$ is a necessary condition, this blackout interval can never be smaller than $\Pi - \Theta$. Then, there will be no advantage in using EDP models for partition interfaces over periodic models. Therefore, we focus on periodic models in the remainder of this paper.

C. Schedulability condition for partitions

Request function. When processes have non-zero offsets, identifying the critical arrival pattern to compute rbf is a non-trivial task. It has been shown that this arrival pattern could occur anywhere in the interval $[0, \mathrm{LCM}]$, where LCM denotes least common multiple of process periods (see [14]). As a result, no closed form expression for rbf is known in this case [6]. Therefore, we now introduce the *request function* ($\mathrm{rf} : \Re \times \Re \to \Re$), which for a given time interval gives the maximum possible amount of resource requested by the partition in that interval. Since rf computes the resource request for a specific time interval as opposed to an interval length, it can be computed without knowledge of the critical arrival pattern. When processes have non-zero jitter in addition to non-zero offsets, we must compute $\mathrm{rf}_{\mathcal{P},i}$ assuming an arrival pattern that results in the maximum higher priority interference for τ_i. The following definition gives this arrival pattern for a job of τ_i with latest release time t, where $t = \mathrm{O}_i + \mathrm{J}_i + x\,\mathrm{T}_i$ for some non-negative integer x.

Definition 1 (Arrival pattern with jitter [25]): Recall that a job of process $\tau = \langle \mathrm{O}, \mathrm{J}, \mathrm{T}, \mathrm{C}, \mathrm{D} \rangle$ is *dispatched* at time instant $x\,\mathrm{T} + \mathrm{O}$ for some non-negative integer x, and can be *released* for execution at any time in the interval $[x\,\mathrm{T} + \mathrm{O}, x\,\mathrm{T} + \mathrm{O} + \mathrm{J}]$. Then, a job of τ_i with latest release time t, incurs maximum interference from higher priority processes in $\mathcal{P}$ whenever, (1) all higher priority processes with dispatch time before t are released at or before t with maximum jitter, and (2) all higher priority processes with dispatch time at or after t are released with zero jitter.
The request function for processes with non-zero offset and jitter values is then given by the following equation.

$$\mathrm{rf}_{\mathcal{P},i}(t_1, t_2) = \sum_{j=1}^{i} \left(\left\lceil \frac{t_2 - \mathrm{O}_j}{\mathrm{T}_j} \right\rceil - \left\lceil \frac{t_1 - \mathrm{O}_j - \mathrm{J}_j}{\mathrm{T}_j} \right\rceil \right) \mathrm{C}_j \quad (6)$$

Schedulability conditions. The following theorem presents exact schedulability conditions for partition $\mathcal{P}$ under periodic resource model ϕ. Due to lack of space, we refer the reader to our technical report for proofs of all theorems presented in this paper [10].

Theorem 2: Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of processes, where for each i, $\tau_i = (\mathrm{O}_i, \mathrm{J}_i, \mathrm{T}_i, \mathrm{C}_i, \mathrm{D}_i)$. Partition $\mathcal{P} = \langle \mathcal{T}, \mathrm{DM} \rangle$ is schedulable using a periodic resource model $\phi = \langle \Pi, \Theta \rangle$ iff $\forall i : 1 \leq i \leq n, \forall t_x$ s.t. $t_x + \mathrm{D}_i - \mathrm{O}_i - \mathrm{J}_i < \mathrm{LCM}_{\mathcal{P}}$ and $t_x = \mathrm{O}_i + \mathrm{J}_i + x\,\mathrm{T}_i$ for some non-negative integer x, $\exists t \in (t_x, t_x + \mathrm{D}_i - \mathrm{O}_i - \mathrm{J}_i]$ such that

$$\mathrm{rf}_{\mathcal{P},i}(0, t) \leq \mathrm{sbf}_\phi(t) \text{ and } \mathrm{rf}_{\mathcal{P},i}(t_x, t) \leq \mathrm{sbf}_\phi(t - t_x) \quad (7)$$

$\mathrm{rf}_{\mathcal{P},i}$ is given by Equation (6) and sbf_ϕ is given by Equation (5). Also, $\mathrm{LCM}_{\mathcal{P}}$ denotes the least common multiple of process periods $\mathrm{T}_1, \ldots, \mathrm{T}_n$.

Proof: To prove that these conditions are sufficient for schedulability of $\mathcal{P}$, we must validate the following statements: (1) it is sufficient to check schedulability of all jobs whose deadlines lie in the interval $[0, \mathrm{LCM}_{\mathcal{P}}]$, and (2) Equation (7) guarantees that the job of τ_i with latest release time t_x, is schedulable using periodic resource model ϕ.

Since $\mathrm{D}_i \leq \mathrm{T}_i$ and $\mathrm{O}_i \leq \mathrm{D}_i$ for all i, no process released before $\mathrm{LCM}_{\mathcal{P}}$ can execute beyond $\mathrm{LCM}_{\mathcal{P}}$ without violating its deadline. Furthermore, dispatch pattern of processes in $\mathcal{P}$ is periodic with period $\mathrm{LCM}_{\mathcal{P}}$. Therefore, it is sufficient to check the schedulability of all jobs in the interval $[0, \mathrm{LCM}_{\mathcal{P}}]$.

We now prove statement (2). Consider the job of τ_i with latest release time t_x. For this job to be schedulable under resource model ϕ, higher priority interference encountered by the job in interval $[t_x, t_x + t)$ must be satisfied by resource model ϕ. This higher priority interference arises from processes released before t_x, as well as from those released at or after t_x. Condition $\mathrm{rf}_{\mathcal{P},i}(t_x, t) \leq \mathrm{sbf}_\phi(t - t_x)$ guarantees that ϕ provides enough supply to satisfy the interference from processes released at or after t_x. To account for the interference from processes released before t_x, we have the second condition, *i.e.*, $\mathrm{rf}_{\mathcal{P},i}(0, t) \leq \mathrm{sbf}_\phi(t)$. This condition ensures that the minimum resource provided by ϕ in an interval of length t, is at least as much as the total higher priority interference up to time t. This proves that these conditions are sufficient for schedulability of partition $\mathcal{P}$.

We now prove that these conditions are also necessary for schedulability of $\mathcal{P}$. For this purpose, observe that $\mathrm{rf}_{\mathcal{P},i}(0, t) \leq \mathrm{sbf}_\phi(t)$ is a necessary condition to guarantee that resource model ϕ satisfies the higher priority interference in interval $[0, t)$. Furthermore, this condition alone is not sufficient, because it does not guarantee that ϕ will provide enough resource in interval $[t_x, t)$. The second condition ensures this property. ■

Periodic resource model based interface for partition $\mathcal{P}$ can be generated using Theorem 2, employing techniques identical to those described at the end of Section III-A. Note that, in this case as well, the theorem needs to be evaluated only at those time instants at which $\mathrm{rf}_{\mathcal{P},i}$ changes. When compared to Theorem 1, this theorem represents a computationally expensive (exponential versus pseudo-polynomial), but more accurate interface generation technique. In fact, for many avionics systems we expect this technique to be computationally efficient as well. For instance, if process periods are harmonic as in many avionics systems (see workloads in the appendix of this technical report [10]), then $\mathrm{LCM}_{\mathcal{P}}$ is simply the largest process period, and our technique has pseudo-polynomial complexity in this case.

[6] $\mathrm{rbf}_{\mathcal{P},i}$ defined in Equation (2) is only an upper bound.

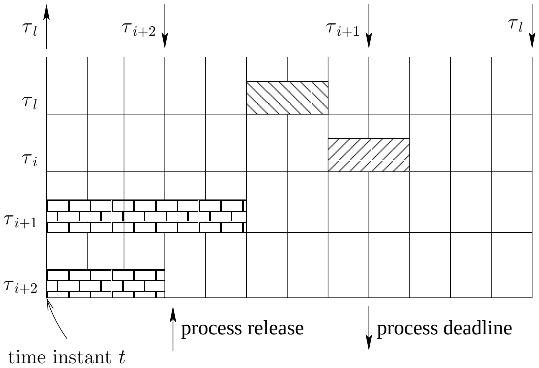


Fig. 5. Illustrative example for $BO_{\mathcal{P},l,i}(t)$

Although Theorem 2 presents an exact schedulability condition for $\mathcal{P}$, it ignores the preemption and blocking overheads incurred by processes in $\mathcal{P}$. Hence, in the following section, we extend our definition of rf to account for these overheads.

Blocking and preemption overheads. Recollect that processes incur blocking overhead because of mutual exclusion requirements modeled using semaphores. Blocking occurs when a lower priority process is executing in a critical section, and a higher priority process cannot preempt this lower priority process. In this case the higher priority process is said to be blocked by the lower priority process, resulting in blocking overheads. Assuming critical sections span entire process executions, two properties of this overhead can be derived immediately: (1) this overhead varies with each job of a process, and (2) any job of a process can be blocked by at most one lower priority process.

Consider a process set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ and partition $\mathcal{P} = \langle \mathcal{T}, \text{DM} \rangle$. We now present an approach to bound the blocking overhead for a job of process τ_l released at time t. Specifically, we compute the bound when this job is blocked by some process having priority lower than that of τ_i, for some $i \geq l$. We assume that all processes with priority lower than τ_i can potentially block this job of τ_l. Our bound is given as

$$BO_{\mathcal{P},l,i}(t) = \max_{k \in [i+1,\ldots,n]} \{\min\{I_k, \mathrm{C}_k\}\}, \tag{8}$$

where I_k is defined as

$$I_k = \begin{cases} 0 & \left\lfloor \frac{t}{\mathrm{T}_k} \right\rfloor \mathrm{T}_k + \mathrm{O}_k \geq t \text{ or } \left\lfloor \frac{t}{\mathrm{T}_k} \right\rfloor \mathrm{T}_k + \mathrm{D}_k \leq t \\ \left\lfloor \frac{t}{\mathrm{T}_k} \right\rfloor \mathrm{T}_k + \mathrm{D}_k - t & \textit{Otherwise} \end{cases}$$

For each process τ_k, we compute its largest interference on the job of τ_l released at time t, and then choose the maximum over all τ_k that have priority lower than τ_i. Any such τ_k released at or before t can block this job of τ_l, and this blocking overhead is at most its worst case execution time. Equation (8) uses this observation to compute the interference from τ_k. Figure 5 gives an illustrative example for this blocking overhead. Let the worst case execution requirement of processes τ_{i+1} and τ_{i+2}, shown in the figure, be 5 time units. Since the deadline of process τ_{i+1} is $t+8$, its interference on the job of τ_l released at t is at most 8. However, its worst case execution requirement is 5, and hence its interference is at most 5 time units. On the other hand, the deadline of process τ_{i+2} is $t+3$, and hence its maximum interference on this job of τ_l is 3 time units.

Note that Equation (8) only gives an upper bound, because the execution of processes τ_j, with $j \leq i$, could be such that no τ_k is able to execute before t. The following equation presents a quantity $BO_{\mathcal{P},l,i}(t_1, t_2)$, which bounds the blocking overhead incurred by all jobs of τ_l released in the interval $[t_1, t_2)$.

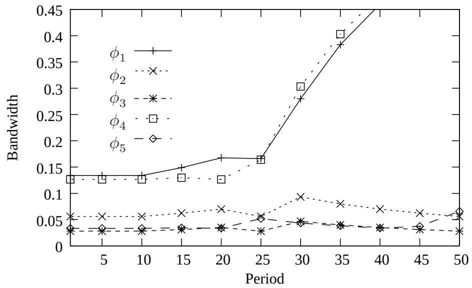
$$BO_{\mathcal{P},l,i}(t_1,t_2) = \sum_{t: t \in [t_1,t_2) \text{ and } \tau_l \text{ released at } t} BO_{\mathcal{P},l,i}(t) \tag{9}$$

When a higher priority process preempts a lower priority process, the context of the lower priority process must be stored for later use. When the lower priority process resumes its execution at some later time instant, this context must be restored. Thus, every preemption results in an execution overhead associated with storing and restoring of process contexts. Many different techniques for bounding this preemption overhead have been proposed in the past (see [23], [11]). Ramaprasad and Mueller [23] have proposed a preemption upper bound for processes scheduled under Rate Monotonic scheduler (RM), and their technique can be extended to other fixed priority schedulers. However, they only present an algorithm to bound the preemptions, but do not give any closed form equations. Easwaran *et. al.* [11] have proposed an analytical upper bound for the number of preemptions under fixed priority schedulers. They presented these bounds for processes with non-zero offset values and zero jitter. These equations can be easily extended to account for jitter in process releases, as well as for blocking overheads. We assume that an upper bound on the number of preemptions is obtained using one such existing technique. Furthermore, we let $PO_{\mathcal{P},i}(t_1, t_2)$ denote this upper bound in the interval $[t_1, t_2)$, for preemptions incurred by processes that have priority at least as much as τ_i. Assuming δ_p denotes the execution overhead incurred by processes for each preemption, request function with blocking and preemption overheads is given as

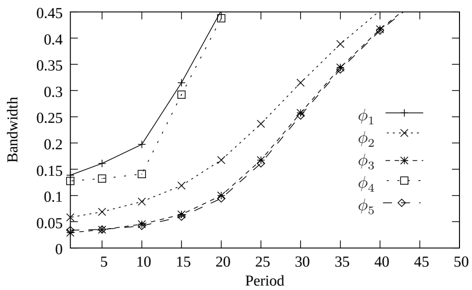
$$\mathrm{rf}_{\mathcal{P},i}(t_1,t_2) = \sum_{j=1}^{i} \left(\left\lceil \frac{t_2 - \mathrm{O}_j}{\mathrm{T}_j} \right\rceil - \left\lceil \frac{t_1 - \mathrm{O}_j - \mathrm{J}_j}{\mathrm{T}_j} \right\rceil \right) \mathrm{C}_j$$
$$+ \delta_p \times PO_{\mathcal{P},i}(t_1,t_2) + \sum_{j=1}^{i} BO_{\mathcal{P},j,i}(t_1,t_2) \tag{10}$$

D. Interface generation for sample workloads

We now demonstrate the effectiveness of our proposed technique using sanitized data sets obtained from an avionics system. These data sets are specified in the appendix in the technical report [10]. There are 7 workloads, where each workload represents a set of partitions scheduled on a single processor. We consider two types of workloads; workloads in which tasks have non-zero offsets but zero jitter (workloads 1 and 2), and workloads in which tasks have non-zero jitter but zero offsets (workloads 3 thru 7). For workloads 1 and 2, Table I in Section III-D1 specifies the total resource utilization of individual partitions ($\sum \frac{\mathrm{C}}{\mathrm{T}}$). For workloads 3 thru 7, Table II in Section III-D2 specifies the resource bandwidth reservations for individual partitions, in addition to total resource utilization. These reservations, currently used by system designers to allocate resources, are computed using the *vmips* parameter

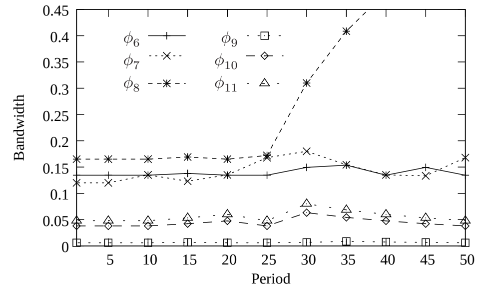


(a) Using Theorem 2

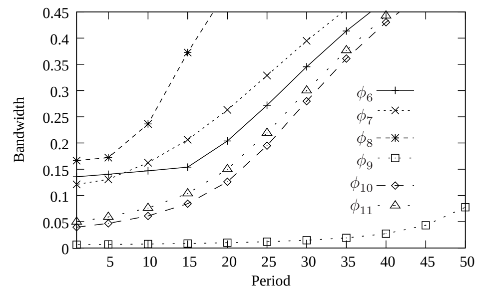


(b) Using approach in [24]

Fig. 6. Interfaces for partitions P1, . . . , P5



(a) Using Theorem 2



(b) Using approach in [24]

Fig. 7. Interfaces for partitions P6, . . . , P11

of the workload specifications (see the appendix of technical report [10]).

We have developed a tool set that takes as input the aforementioned avionics hierarchical systems, and generates as output resource model based interfaces for them. In the following two sections we present the results generated using this tool set.

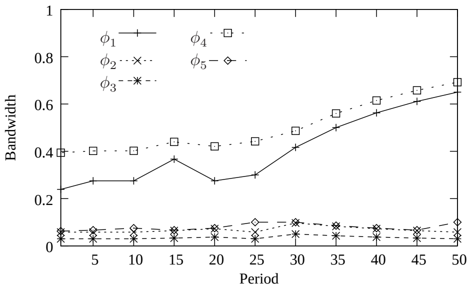
Partition	Utilization	Partition	Utilization
$P1$	0.134	$P6$	0.12
$P2$	0.056	$P7$	0.1345
$P3$	0.028	$P8$	0.165
$P4$	0.1265	$P9$	0.006
$P5$	0.0335	$P10$	0.038
		$P11$	0.048

TABLE I
WORKLOADS 1 AND 2

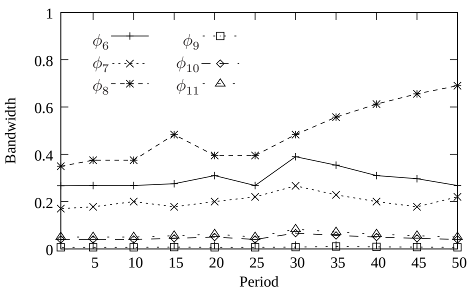
1) Workloads with non-zero offsets: In this section, we consider workloads 1 and 2. Firstly, we compare our proposed approach with the existing well known compositional analysis technique based on periodic resource models [24]. We assume that this technique uses Theorem 1 to generate periodic resource model based partition interfaces, and therefore ignores process offsets. This approach does not account for preemption and blocking overheads incurred by processes. Hence to ensure a fair comparison, we ignore these overheads when computing interfaces using our approach as well. In Figures 6(a) and 7(a), we have plotted the resource bandwidths of interfaces obtained using our approach (Theorem 2). We have plotted these bandwidths for period values 1 and multiples of 5 up to 50. Note that since sbf_{ϕ} defined in Equation (5) is a linear function of capacity Θ, there is no need to use a linear lower bound like the one used in [24]. Similarly, we also obtained partition interfaces using Theorem 1 as discussed above, and their resource bandwidths are plotted in Figures 6(b) and 7(b).

As can be seen from these plots, interfaces obtained using our approach have a much smaller resource bandwidth when compared to those obtained using the existing technique. This gain in efficiency is because of two reasons: (1) we use a tighter sbf in Theorem 2 when compared to existing approach, and (2) existing approach ignores process offsets, and hence generates pessimistic interfaces. Although this is only an illustrative example, it is easy to see that the advantages of our interface generation technique hold in general. From the plots in Figures 6(a) and 7(a) we can also see that for some period values, bandwidths of our periodic resource models are equal to the utilization of corresponding partitions. Since utilization of a partition is the minimum possible bandwidth of a resource model that can schedule the partition, our approach generates optimal resource models for these periods. In these plots it can also be observed that the bandwidth increases sharply beyond a certain period. For interfaces ϕ_1, ϕ_4, and ϕ_8 corresponding to partitions $\mathcal{P}_1, \mathcal{P}_4$, and $\mathcal{P}_8$, respectively, the bandwidth increases sharply beyond period 25. This increase can be attributed to the fact that in these partitions the smallest process period is also 25. In our examples, since smallest process period corresponds to the earliest deadline in a partition, resource models with periods greater than this smallest value require larger bandwidth to schedule the partition.

Finally, we also generated partition interfaces using Theorem 2, taking into account preemption and blocking overheads. The resource bandwidth of these interfaces are plotted in Figures 8(a) and 8(b). For preemption overhead we assumed that the overhead for each preemption δ_p is 0.1, and that



(a) Partitions P1, . . . , P5



(b) Partitions P6, . . . , P11

Fig. 8. Partition interfaces with blocking and preemption overheads

Partition name	Utilization	Reserved	Computed	% Increase
Workload 3				
PART16 ID=16	0.01965	0.04505	0.0246	83.1%
PART29 ID=29	0.199415	0.37669	0.3735	0.9%
PART35 ID=35	0.05168	0.22185	0.0717	209.4%
PART20 ID=20	0.035125	0.09798	0.0589	66.3%
PART32 ID=32	0.033315	0.08164	0.0781	4.5%
PART36 ID=36	0.045	0.11036	0.12	−8%
PART33 ID=33	0.0379	0.09178	0.0579	58.5%
PART34 ID=34	0.04764	0.10755	0.0676	59.1%
PART17 ID=17	0.00408	0.01126	0.0082	37.3%
PART31 ID=31	0.00684	0.01689	0.0137	23.3%
Workload 4				
PART30 ID=30	0.11225	0.23086	0.169	36.6%
PART16 ID=16	0.01965	0.04505	0.0246	83.1%
PART20 ID=20	0.035125	0.09797	0.0589	66.3%
PART17 ID=17	0.00408	0.01126	0.0082	37.3%
PART26 ID=26	0.13496	0.44932	0.2538	77%
PART27 ID=27	0.02784	0.06869	0.0478	43.7%
PART28 ID=28	0.0552	0.12106	0.0752	61%
Workload 5				
PART15 ID=15	0.5208	0	0.5224	
PART13 ID=13	0.01126	0.03378	0.0163	107.2%
PART12 ID=12	0.0050	0.01126	0.02	−43.7%
Workload 6				
PART16 ID=16	0.01965	0.04505	0.0246	83.1%
PART19 ID=19	0.14008	0.32939	0.2284	44.2%
PART21 ID=21	0.12751	0.30011	0.2667	12.5%
PART22 ID=22	0.13477	0.31137	0.2631	18.3%
PART17 ID=17	0.00408	0.01126	0.0082	37.3%
Workload 7				
PART45 ID=45	0.00325	0.02815	0.01	181.5%

TABLE II
BANDWIDTHS FOR WORKLOADS 3 THRU 7

every job of a process preempts some lower priority process. Blocking overhead was computed using the upper bound given in Equation (9). As expected, resource bandwidths of these interfaces are significantly higher in comparison to the bandwidths in Figures 6(a) and 7(a) [7]. Since our preemption and blocking overheads are only upper bounds and not necessarily tight, the minimum bandwidths of resource models that can schedule these partitions lie somewhere in between the two plots.

2) Workloads with non-zero jitter: In this section, we consider workloads 3 thru 7. Since these workloads have zero offsets, we used Theorem 1 to generate periodic resource model based partition interfaces. In this theorem, we used sbf given by Equation (5), and interface periods are as specified by the *min-period* and *max-period* fields of component tags [8]. For preemption overheads we assumed that the overhead for each preemption δ_p is 0.1, and that every job of a process preempts some lower priority process. For blocking overheads we assumed that every lower priority process can block the process under consideration, up to its worst case execution time. Consider the process set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ and partition $\mathcal{P} = \langle \mathcal{T}, \mathrm{DM} \rangle$. Then, for a process $\tau_l \in \mathcal{T}$, its blocking overhead is equal to $\max_{k>l}\{\mathrm{C}_k\}$.

We now compare the bandwidth of generated interfaces with the reserved bandwidth of partitions. Table II lists the following four parameters for each partition in workloads 3 thru 7: (1) Total utilization of the partition ($\sum \frac{\mathrm{C}}{\mathrm{T}}$), (2) Reserved bandwidth, (3) Interface bandwidth computed as described above, and (4) Percentage increase in bandwidth ($\frac{\text{reserved} - \text{computed}}{\text{computed}} \times 100$). As can be seen from this table, bandwidths of partition interfaces generated using our technique are significantly smaller than reserved bandwidths of partitions. However, when generating partition interfaces, we ignore the resource requirements of aperiodic processes in partitions. These aperiodic processes are identified by a period value of zero in the workload specifications in the technical report. For example, they are present in partition "PART26 ID=26" of workload 4 and partition "PART22 ID=22" of workload 6. Since the workloads do not specify any deadlines for these processes (they execute as background processes in ARINC-653), we cannot determine the resource utilization of these processes. Then, one may argue that the difference in reserved bandwidth and bandwidth computed by our technique, is in fact used by aperiodic processes. Although this can be true, our results show that even for partitions with no aperiodic processes, there are significant savings using our technique.

IV. PARTITION SCHEDULING

Let the partition set $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be scheduled on an uniprocessor platform under DM scheduler. Furthermore, let each partition $\mathcal{P}_i$ be represented by a periodic resource model based interface $\phi_i = \langle \Pi_i, \Theta_i \rangle$ as described in Section III. Without loss of generality we assume that $\Pi_1 \leq \ldots \leq \Pi_n$. To schedule these interfaces on the uniprocessor platform, we must transform each resource model into a task that the

[7] Y-axis in Figures 8(a) and 8(b) ranges from 0 to 1, whereas in Figures 6(a) and 7(a) it ranges from 0 to 0.45.

[8] Note that *min-period* = *max-period* in all the component tags in workloads 3 thru 7.

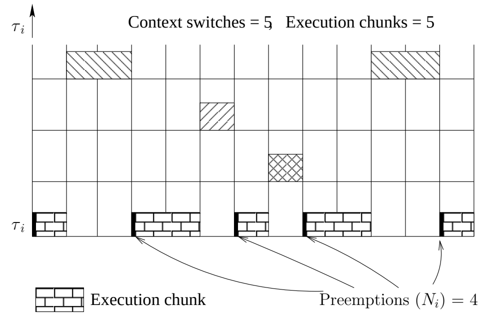


Fig. 9. Preemption count terminology

higher level DM scheduler can use. For this purpose, we use the transformation which for interface ϕ_i generates the process $\tau_i = (0, 0, \Pi_i, \Theta_i, \Pi_i)$. It has been shown that this transformation is both necessary and sufficient w.r.t. resource requirements of ϕ_i [24].

If each partition interface is transformed as above, then processes in the resulting set $(\tau_1, \ldots, \tau_n)$ have implicit deadlines, zero offset values, and harmonic periods (partition periods are harmonic). Liu and Layland have shown that DM is an optimal scheduler for such processes [19]. In the following section we present a technique to count the number of preemptions incurred by this process set. The partition level schedule can then be generated after adjusting execution requirements of $\tau_1, \ldots, \tau_n$ to account for preemption overheads.

A. Partition level preemption overhead

Preemption overhead for partitions represented as processes, can be computed using the upper bounds described in Section III. However, as described in the previous section, these processes are scheduled under DM, and have harmonic periods, implicit deadlines, and zero offset and jitter values. For such a process set, it is easy to see that every job of each process executes in the same time instants relative to its release time (see Figure 4). Therefore, every job of a process is preempted an identical number of times. For this case, we now develop an analytical technique to compute the exact number of preemptions.

Consider the process set $\tau_1, \ldots, \tau_n$ defined in the previous section. For each i, let N_i denote the number of preemptions incurred by each job of τ_i. We first give an upper bound for N_i, and later show how to tighten this bound. For this upper bound, we assume that the number of preemptions $N_1, \ldots, N_{i-1}$ for processes $\tau_1, \ldots, \tau_{i-1}$, respectively, are known. We also assume that the worst case execution requirements of these processes are adjusted to account for preemption overheads. Then, the following iterative equation gives an upper bound for N_i.

$$N_i^{(k)} = \left\lceil \frac{\Theta_i^{(k)}}{\Pi_{i-1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} \Theta_j} \right\rceil \left(\frac{\Pi_{i-1}}{\Pi_1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} N_j \right) - 1 \tag{11}$$

In this equation we assume $\Theta_i^{(0)} = \Theta_i$ and $\Theta_i^{(k)} = \Theta_i + N_i^{(k-1)} \delta_p + \delta_p$, where δ_p denotes the execution overhead for each preemption. $N_i^{(k)}$ ignores the preemption incurred by process τ_i at the start of its execution, and hence the additional δ_p in capacity adjustment (see Figure 9). Then, the upper bound for N_i is given by that value of $N_i^{(k)}$ for which $N_i^{(k)} = N_i^{(k-1)}$.

Theorem 3: Let N_i^* denote the value of $N_i^{(k)}$ in Equation (11) such that $N_i^{(k)} = N_i^{(k-1)}$. Then $N_i^* \geq N_i$.

In the k^{th} iteration, given $\Theta_i^{(k)}$, Equation (11) computes the number of dispatches of process τ_{i-1} that occur before the execution of $\Theta_i^{(k)}$ units of τ_i. We then determine the number of preemptions incurred by τ_i within the execution window of each these dispatches of τ_{i-1}. Use of ceiling function in the equation gives an upper bound to this number. To determine the number of preemptions within each execution window of τ_{i-1}, Equation (11) computes the number of execution chunks of τ_i in each window. Each set of consecutive execution units of a process in a schedule is a single execution chunk [9]. The maximum possible number of chunks is given by $\frac{\Pi_{i-1}}{\Pi_1}$. However, since higher priority processes also execute in this window, τ_i need not have so many execution chunks. Therefore, we subtract the execution chunks of higher priority processes from this maximum possible number. We use N_j, an upper bound, for the number of execution chunks of higher priority process τ_j. See proof of Theorem 3 in the technical report for a full explanation of Equation (11).

Since $\Theta_i^{(k)}$ is non-decreasing and cannot be greater than Π_i, this iterative computation must terminate and has pseudo-polynomial complexity. This computation only gives an upper bound for N_i due to two reasons: (1) the ceiling function, and (2) use of N_j as the count for execution chunks of process τ_j. In fact, Equation (11) cannot be used to upper bound N_i, because it assumes knowledge of preemption counts $N_1, \ldots, N_{i-1}$. We now present a technique that overcomes these shortcomings. In particular, we modify Equation (11) as follows:

- We replace ceiling with the floor function, and add a separate expression that counts preemptions in the last execution window of τ_{i-1}.
- We replace N_j in the equation with a quantity I_j, which is either N_j+1 or N_j, depending on whether the response time of τ_j coincides with a release of τ_1.

Let $N_i^{(k)'}$ denote the preemption count for τ_i in the last execution window of τ_{i-1}, when $\Theta_i^{(k)}$ is the execution requirement of τ_i. Then, N_i is given by the following iterative equation.

$$N_i^{(k)} = \left\lfloor \frac{\Theta_i^{(k)}}{\Pi_{i-1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} \Theta_j} \right\rfloor \left(\frac{\Pi_{i-1}}{\Pi_1} - \sum_{j=1}^{i-1} \frac{\Pi_{i-1}}{\Pi_j} I_j \right) + N_i^{(k)'} - 1 \tag{12}$$

In this equation we assume $\Theta_i^{(0)} = \Theta_i$ and $\Theta_i^{(k)} = \Theta_i + N_i^{(k-1)} \delta_p + \delta_p$. Also, N_i is given by that value of $N_i^{(k)}$ for which $N_i^{(k)} = N_i^{(k-1)}$. We now give equations to compute the

[9]Note that the number of execution chunks is always one more than the number of preemptions encountered by the process.

two unknown quantities, I_j and $N_i^{(k)'}$ in the above equation.

$$I_j = \begin{cases} N_j + 1 & \left\lceil \frac{\mathrm{R}_j}{\Pi_1} \right\rceil = \left\lfloor \frac{\mathrm{R}_j}{\Pi_1} \right\rfloor \\ N_j & \text{Otherwise} \end{cases}$$

Here R_j denotes the worst case response time of process τ_j. Since $j \in [1, \ldots, i-1]$, N_j is known and therefore R_j can be computed. $N_i^{(k)'}$ is given by the following equation.

$$N_i^{(k)'} = \left\lceil \frac{\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}}{\Pi_1} \right\rceil - \sum_{j=2}^{i-1} \left\lceil \frac{\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}}{\Pi_j} \right\rceil I_j \qquad (13)$$

In this equation $\mathrm{R}_i^{(k)}$ denotes the response time of τ_i with execution requirement $\Theta_i^{(k)}$, and $T_{i-1}^{(k)}$ is the time of last dispatch of τ_{i-1}. $\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}$ gives the total time taken by τ_i to execute in the last execution window of τ_{i-1}. This, along with the higher priority interference in the window, gives $N_i^{(k)'}$. The following theorem then observes that the preemption count generated using Equation (12) is equal to N_i.

Theorem 4: Let N_i^* denote the value of $N_i^{(k)}$ in Equation (12) such that $N_i^{(k)} = N_i^{(k-1)}$. Then $N_i^* = N_i$.

In this iterative procedure as well, $\Theta_i^{(k)}$ is non-decreasing and cannot be greater than Π_i. Therefore, the computation is of pseudo-polynomial complexity in the worst case. One may argue that the exact preemption count can also be obtained by simulating the execution of processes. Since process periods are harmonic, LCM is simply the largest process period, and therefore the simulation also runs in pseudo-polynomial time. However, in safety critical systems such as avionics, it is often required that we provide analytical guarantees for correctness. The iterative computation presented here serves this purpose.

Thus, each process τ_i can be modified to account for preemption overhead and is specified as $\tau_i = (0, 0, \Pi_i, \Theta_i + (N_i + 1)\delta_p, \Pi_i)$. If the resulting process set $\{\tau_1, \ldots, \tau_n\}$ is schedulable[10], then using Theorems 2 and 4 we get that the underlying partitions can schedule their workloads.

V. Conclusions

In this paper we presented ARINC-653 standards for avionics real-time OS, and modeled it as a two level hierarchical system. We extended existing resource model based techniques to handle processes with non-zero offset values. We then used these techniques to generate partition level schedules. Design of real-time systems in modern day air-crafts is done manually through interactions between application vendors and system designers. Techniques presented in this paper serve as a platform for principled design of partition level schedules. They also provide analytical correctness guarantees, which can be used in system certification.

[10] Liu and Layland have given response time based schedulability conditions for this case [19].

References

[1] Green Hills Software, ARINC 653 partition scheduler. In *www.ghs.com/products/safety_critical/arinc653.html*.

[2] Windriver, platform for ARINC 653. In *www.windriver.com/products/platforms/safety_critical/*.

[3] Avionics application software standard interface: Part 1 - required services (arinc specification 653-2). Technical report, Avionics Electronic Engineering Committee (ARINC), March 2006.

[4] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *EMSOFT*, 2004.

[5] 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.

[6] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Using harmonic task-sets to increase the schedulable utilization of cache-based preemptive real-time systems. In *RTCSA*, 1996.

[7] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *RTSS*, 2005.

[8] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *RTSS*, 2006.

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

[10] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A compositional framework for avionics (arinc-653) systems. Technical Report MS–CIS–09–04, University of Pennsylvania, 2009. Available at http://repository.upenn.edu/cis_reports/898/.

[11] A. Easwaran, I. Shin, I. Lee, and O. Sokolsky. Bounding preemptions under EDF and RM schedulers. Technical Report MS–CIS–06–06, University of Pennsylvania, USA, 2006.

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

[13] N. Fisher, M. Bertogna, and S. Baruah. The design of an edf-scheduled resource-sharing open environment. In *RTSS*, 2007.

[14] J. Goossens. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints*. PhD thesis, Universit Libre de Bruxelles, 1999.

[15] L. Kinnan, J. Wlad, and P. Rogers. Porting applications to an arinc 653 compliant ima platform using vxworks as an example. In *Proceedings of the 23rd Digital Avionics Systems Conference*, 2004.

[16] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou. Scheduling tool and algorithm for integrated modular avionics systems. In *Proceedings of 19th Digital Avionics Systems Conference*, 2000.

[17] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, pages 237–250, 1982.

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

[19] C. Liu and J. Layland. Scheduling algorithms for multi-programming in a hard-real-time environment. *Journal of the ACM*, 20(1):46 – 61, 1973.

[20] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *RTSS*, 2005.

[21] A. K. Mok, D.-C. Tsou, and R. C. M. de Rooij. The msp.rtl real-time scheduler synthesis tool. In *RTSS*, 1996.

[22] M. D. Natale and J. Stankovic. Dynamic end-to-end guarantees in distributed real-time systems. In *RTSS*, 1994.

[23] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *RTSS*, 2006.

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

[25] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report: YCS 221, Dept. of Computer Science, University of York, York, England, January 1994.

[26] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40:171–134, 1994.