A Compositional Scheduling Framework for Digital Avionics Systems

Arvind Easwaran

Insup Lee

Oleg Sokolsky

Steve Vestal

HURRAY-TR-090905

Version: 0

Date: 09-11-2009

A Compositional Scheduling Framework for Digital Avionics Systems

Arvind Easwaran[1] and Insup Lee[2] and Oleg Sokolsky[2] and Steve Vestal[3]

[1]IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: aen@isep.ipp.pt

http://www.hurray.isep.ipp.pt

[2]Department of CIS

University of Pennsylvania

PA, 19104, USA

E-mail: {flee,sokolskyg}@cis.upenn.edu

[3]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 partitionconsists 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. In contrast, the technique proposed in this paper is a principled approach for scheduling ARINC-653 partitions and therefore should facilitate system integration.

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. In contrast, the technique proposed in this paper is a principled approach for scheduling ARINC-653 partitions and therefore should facilitate system integration.**

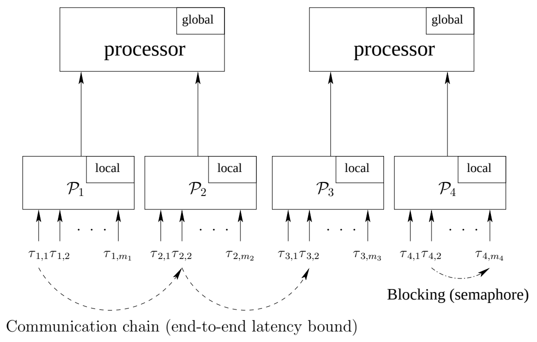

global
processor
global
processor
local
$\mathcal{P}_1$
local
$\mathcal{P}_2$
local
$\mathcal{P}_3$
local
$\mathcal{P}_4$
$\tau_{1,1}$ $\tau_{1,2}$ τ_{1,m_1}
$\tau_{2,1}$ $\tau_{2,2}$ τ_{2,m_2}
$\tau_{3,1}$ $\tau_{3,2}$ τ_{3,m_3}
$\tau_{4,1}$ $\tau_{4,2}$ τ_{4,m_4}
Blocking (semaphore)
Communication chain (end-to-end latency bound)

Fig. 1. Scheduling hierarchy for partitions

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 [1] (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. 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 overheads* (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*.

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 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. Further, these scheduling parameters must

This research was supported in part by AFOSR FA9550-07-1-0216, NSF CNS-0509327, NSF CNS-0720703, ONR MURI N00014-07-1-0907, NSF CNS-0721541 and NSF CNS-0720518. This work was also partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the European Commission through grant ArtistDesign ICT-NoE-214373.

Arvind Easwaran contributed to this work when he was a PhD student at the University of Pennsylvania, USA.

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. Further, 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 *partition interfaces* which are abstract representations of the resource demand of processes within partitions.

Partition workloads can be abstracted into interfaces using existing compositional techniques [2], [3], [4], [5]. 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 [2], [4], bounded-delay [3] and Explicit Deadline Periodic (EDP) [5], 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 [6], [7]). 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 [2] 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 [8]. 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[1]. 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 D ($\leq$ 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 $[x\,\mathrm{T} + \mathrm{O}, x\,\mathrm{T} + \mathrm{O} + \mathrm{J}]$. Each job requires C units of processing capacity within D time units from its dispatch. For such a process it is reasonable to assume that $\mathrm{O} \leq \mathrm{D}$ [6]. Further, we denote as $\langle \{\tau_1, \ldots, \tau_n\}, \mathrm{DM} \rangle$ a partition $\mathcal{P}$ that comprises of processes $\tau_1, \ldots, \tau_n$ and uses DM scheduler. Without loss of generality we assume that in this partition τ_i has higher priority than τ_j for all $i < j$.

In addition to the restrictions specified so far, we make further assumptions for the system described herein. These assumptions have been verified to be true in digital avionics systems. For example, see the avionics workloads given in the appendix of this technical report [9]. (1) We assume that the processes within a partition, and hence the partition itself, cannot be distributed over multiple processors. (2) We assume that periods of partitions that are scheduled on the same processor are *harmonic*[2]. Note that this assumption does not prevent processes from having non-harmonic periods. (3) We assume that processes in one partition cannot cause blocking to processes in another partition. This follows from the fact that mutual exclusion based on semaphores requires use of shared memory, which is only possible 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 provide the partitions. Although many different ARINC-653 platforms exist (see [10], [11]), there is little work on automatic scheduling of partitions [12], [13], [14]. Kinnan *et al.* [12] only provide preliminary heuristic guidance, and the other studies [13], [14] use constraint-based approaches to look

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

[2]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.

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 [4], [5], [2]. However these studies do not consider dependencies between and within partitions. But such dependencies in hierarchical systems have been addressed in other studies [15], [16], [17], [18], [19], [20]. Almeida and Pedreiras [15] have presented compositional analysis techniques for the case when processes in partition workload have jitter in their releases. Davis and Burns [16] have extended this technique to consider release jitter as well as preemption overheads. Various resource-sharing protocols (HSRP [18], SIRAP [19], BROE [20]) 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 [17] have also developed compositional analysis techniques in the presence of partition dependencies. Although one of their dependency models (real-time workshop) is similar to the dependency constraints that we consider here, it is more restrictive in that periods of dependent processes are required to be harmonic.

Mataix *et al.* [21] 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 resource model based interface for a partition $\mathcal{P} = \langle\{\tau_1, \ldots, \tau_n\}, \text{DM}\rangle$. We assume that $\Pi_\mathcal{P}$ denotes the interface period specified by the system designer for $\mathcal{P}$. We first briefly discuss the 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 processing capacity required to meet all its temporal deadlines ($\mathrm{rbf} : \Re \rightarrow \Re$). 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 $\mathrm{rbf}_{\mathcal{P},i}(t)$ the request bound function of process τ_i in partition $\mathcal{P}$ for a time interval length t. Then Equation (1) gives $\mathrm{rbf}_{\mathcal{P},i}$ assuming that jitter and offset for all the processes are zero [4].

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

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 [22]. In this arrival pattern each higher priority process is released simultaneously with the process under consideration, incurring maximum possible jitter. All future jobs of these higher priority processes are released as soon as possible, *i.e.*, they incur zero jitter. Further, 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, $\mathrm{rbf}_{\mathcal{P},i}$ can be specified as

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

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 in every Π units of time can be represented using the periodic resource model $\phi = \langle\Pi, \Theta\rangle$ [4]. 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 $\eta = \langle\Pi, \Theta, \Delta\rangle$ [5]. In both these models, $\frac{\Theta}{\Pi}$ represents the 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 ($\mathrm{sbf} : \Re \rightarrow \Re$). Given a resource model R and time interval length t, $\mathrm{sbf}_R(t)$ 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 the *blackout intervals* for periodic and EDP models respectively. These functions are also plotted in Figure 2. As shown in the figure, after the blackout interval, the models provide Θ units of resource in every Π time units.

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

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

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 [4], [5]. These conditions can be easily extended for processes with non-zero jitter and is presented below.

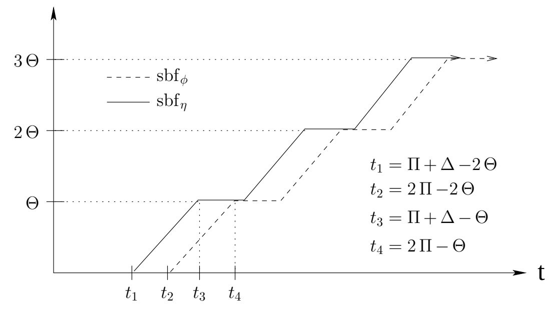


Fig. 2. sbf for periodic and EDP models

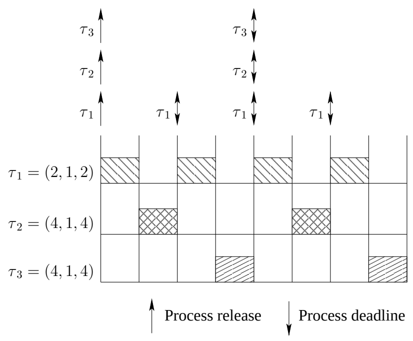


Fig. 3. Tasks with harmonic periods

Theorem 1: A partition $\mathcal{P} = \langle\langle \tau_1 = (0, \mathrm{J}_1, \mathrm{T}_1, \mathrm{C}_1, \mathrm{D}_1), \ldots, \tau_n = (0, \mathrm{J}_n, \mathrm{T}_n, \mathrm{C}_n, \mathrm{D}_n)\rangle, \textsc{dm}\rangle$ is schedulable over a periodic or EDP resource model R iff

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

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 [4] 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.* [5] have presented a technique that uses this theorem to compute a resource model having the 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 (not necessarily tight) 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 the 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 the periodic resource model and then extend Theorem 1.

B. sbf *under harmonic interface periods*

In the technique described in [4], a periodic interface $\phi = \langle \Pi, \Theta \rangle$ is transformed into a periodic task $\tau_\phi = (\Pi, \Theta, \Pi)$[3] before it is presented to the global scheduler. Note that the period of model ϕ and task τ_ϕ are identical and the period of task τ_ϕ is identical to its deadline. In ARINC-653, this means the partitions that are scheduled on a processor are in fact abstracted into periodic tasks with harmonic periods. When such implicit deadline (period = deadline) 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 3 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 the 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 3, 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 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 an EDP resource model $\eta = \langle \Pi, \Theta, \Delta \rangle$, the blackout interval in sbf_η is $\Pi + \Delta - 2\,\Theta$ [5]. Since $\Delta \ge \Theta$ 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.

[3]This task is equivalent to the constrained deadline periodic task $(0, 0, \Pi, \Theta, \Pi)$, i.e., it has zero jitter and offset. For convenience of presentation, we ignore the leading zeroes in the notation.

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 the least common multiple of process periods (see [23]). As a result no closed form expression for rbf is known in this case. Therefore we now introduce the *request function* ($\mathrm{rf} : \Re \times \Re \rightarrow \Re$), 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 [6]): Recall that a job of process $\tau = (\mathrm{O}, \mathrm{J}, \mathrm{T}, \mathrm{C}, \mathrm{D})$ 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.

To compute $\mathrm{rf}_{\mathcal{P},i}(t_1, t_2)$ based on the above arrival pattern, we do the following for each higher priority process. We first count the number of jobs that can be released up to time t_2, assuming zero jitter. We then subtract from this count the number of jobs that can be released up to time t_1, assuming maximum jitter. Thus

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

Schedulability conditions. The following theorem presents exact schedulability conditions for partition $\mathcal{P}$ under periodic resource model ϕ.

Theorem 2: Let $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ denote a set of processes and $\mathcal{P} = \langle \mathcal{T}, \mathrm{DM} \rangle$ denote a partition, where for each i, $\tau_i = (\mathrm{O}_i, \mathrm{J}_i, \mathrm{T}_i, \mathrm{C}_i, \mathrm{D}_i)$. Also, let $\mathrm{LCM}_{\mathcal{P}}$ denote the least common multiple of process periods $\mathrm{T}_1, \ldots, \mathrm{T}_n$. $\mathcal{P}$ 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

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

Proof: To prove that these conditions are sufficient for the schedulability of $\mathcal{P}$, we must validate the following statements: (1) it is sufficient to check the 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. Further, 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 the 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 the schedulability of partition $\mathcal{P}$.

We now prove that these conditions are also necessary for the 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 $[0, t)$. Further, this condition alone is not sufficient, because it does not guarantee that ϕ will provide enough resource in the 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 technique represents a computationally expensive (exponential versus pseudo-polynomial) but more accurate interface generation process. 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 [9]), then $\mathrm{LCM}_{\mathcal{P}}$ is simply the largest process period and our technique has pseudo-polynomial complexity in this case.

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. Recall that processes incur blocking overhead because of mutual exclusion requirements modeled using semaphores. Blocking occurs when a job of a lower priority process is executing in a critical section, and a job of a higher priority process cannot preempt this lower priority job. In this case the higher priority job is said to be blocked by the lower priority job, resulting in blocking overheads. In this paper we assume that critical sections span entire job executions and they are non-preemptible. That is a job locks each shared data it wants to access at the beginning of its execution and releases these locks only when it finishes its entire execution. Two properties of the blocking overhead can then 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 job.

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 $\tau_l \in \mathcal{T}$ released at time t. Specifically, we compute the bound when this job is blocked by some job having priority lower than that of process τ_i, for some $i \geq l$. For each process τ_k ($k > i$), we compute its largest interference on the job of τ_l released at time t (I_k), and then choose the maximum over all such processes. Suppose J denotes a job of process τ_k that is released before time t but has a deadline after t. If no such job exists, then interference I_k is zero. Otherwise, we set I_k equal to the maximum time for which J can execute after time t without missing its deadline.

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

Since the maximum time for which a job of τ_k can block a higher priority job is upper bounded by its worst case execution time C_k, the blocking overhead for a job of τ_l released at time t is given as

$$BO_{\mathcal{P},l,i}(t) = \max_{k>i} \{\min\{I_k, \mathrm{C}_k\}\}, \tag{8}$$

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 the storing and restoring of process contexts. Many different techniques for bounding this preemption overhead have been proposed in the past. For example, Easwaran *et al.* [24] 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. Further, 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

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

The blocking overhead in the above equation considers the total blocking incurred by all processes τ_j ($j \leq i$) from processes having priority lower than τ_i. This in fact represents the total lower priority workload that can potentially execute at a priority higher than that of τ_i.

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 [9]. 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 offset but zero jitter (workloads 1 and 2), and workloads in which tasks have non-zero jitter but zero offset (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 the system designers to allocate resources, are computed using the *vmips* parameter of the workload specifications[4].

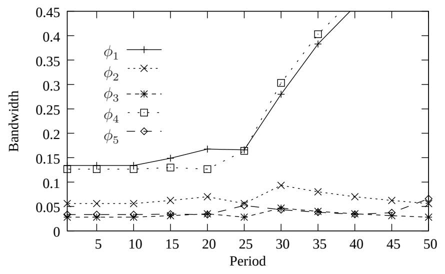
We have developed a tool set that takes as input the aforementioned workloads, and generates as output resource model based interfaces for them. In the following two sections we present the results generated using this tool set.

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

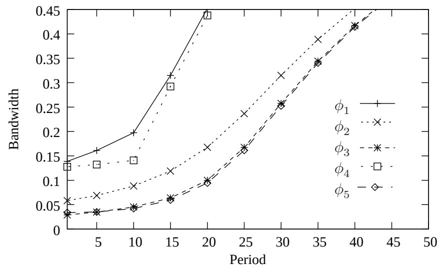
TABLE I
WORKLOADS 1 AND 2

1) Workloads with non-zero offset: 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 the periodic resource model [4]. We assume that this technique uses Theorem 1 to generate periodic resource model based partition interfaces and therefore

[4]See the appendix of the technical report for details on how the reservations are computed.

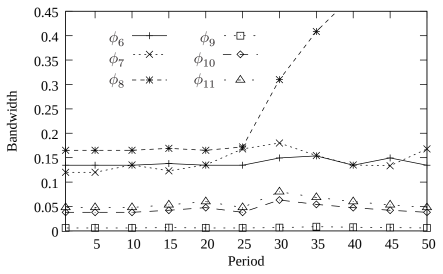


(a) Using Theorem 2

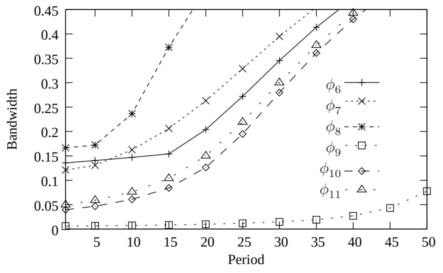


(b) Using approach in [4]

Fig. 4. Interfaces for partitions P1, . . . , P5



(a) Using Theorem 2



(b) Using approach in [4]

Fig. 5. Interfaces for partitions P6, . . . , P11

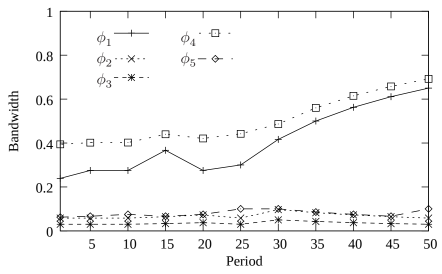
ignores process offsets. This approach does not account for the preemption and blocking overheads incurred by processes. Hence to ensure a fair comparison, we ignore these overheads when computing interfaces using our approach as well. In Figures 4(a) and 5(a), we have plotted the resource bandwidths 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 [4]. Similarly, we also obtained partition interfaces using Theorem 1 as discussed above, and their resource bandwidths are plotted in Figures 4(b) and 5(b). Although partition (and therefore interface) periods are fixed by the system designers, we have plotted the resource bandwidths for different period values. This is to illustrate that our results in this section, that of comparison of the two approaches, are independent of partition periods.

As can be seen from these plots, interfaces obtained using our approach have a much smaller resource bandwidth on an average 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 the existing approach, and (2) the existing approach ignores process offsets and hence generates pessimistic interfaces. These advantages of our approach over the existing one hold in general, and therefore the illustrative results of this section are also expected to hold for other workloads in the avionics domain. From the plots in Figures 4(a) and 5(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.

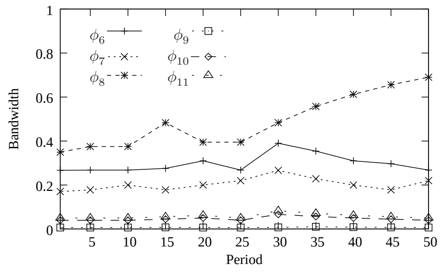
Finally, we also generated partition interfaces using Theorem 2, taking into account preemption and blocking overheads. The resource bandwidth of these interfaces are plotted in Figures 6(a) and 6(b). For preemptions we assumed that the overhead of each preemption (δ_p) is 0.1, and that every job of a process preempts some lower priority job. This bound on δ_p is consistent with the observed preemption overhead on real avionics workloads (*i.e.*, 10% of one unit of execution on an average). 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 4(a) and 5(a) [5]. 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 offset, we used Theorem 1 to generate periodic resource model based partition interfaces. In this theorem we used sbf given

[5] Y-axis in Figures 6(a) and 6(b) ranges from 0 to 1, whereas in Figures 4(a) and 5(a) it ranges from 0 to 0.45.



(a) Partitions P1, . . . , P5



(b) Partitions P6, . . . , P11

Fig. 6. Partition interfaces with blocking and preemption overheads

by Equation (5) and interface periods are set equal to the corresponding partition periods given in the workload specifications. Preemption overheads are assumed to be identical to those described in the previous section. For blocking overhead of a process τ_l we use $\max_{t:\tau_l \text{ is released at } t} BO_{\mathcal{P},l,l}(t)$. This is in fact equal to $\max_{k>l}\{\mathrm{C}_k\}$ because the process has zero offset.

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 ($100 \times$ (reserved $-$ computed)/computed). As can be seen from this table, bandwidths of partition interfaces generated using our technique are significantly smaller than the reserved bandwidths of partitions on an average. 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. 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 is possible, our results show that even for partitions with no aperiodic processes there are significant savings using our technique. The bandwidth computed using our technique is higher than the reserved bandwidth for two partitions ("PART36 ID=36" and "PART12 ID=12"). Without more information on how

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

the reserved bandwidths are computed, the only reasonable explanation for such exceptions is that our estimates on preemption and blocking overheads are far higher than their actual values in these partitions.

IV. PARTITION SCHEDULING

Let the partition set $\mathcal{P}_1, \dots, \mathcal{P}_n$ be scheduled on an uniprocessor platform under DM scheduler. Further, 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 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 with respect to the resource requirements of ϕ_i [4].

If each partition interface is transformed as above, then processes in the resulting set $(\tau_1, \dots, \tau_n)$ have implicit deadlines, zero offset and harmonic periods (partition periods are harmonic). Liu and Layland have shown that DM is an optimal scheduler for such processes [25]. 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 the execution requirements of $\tau_1, \dots, \tau_n$ to account for preemption overheads.

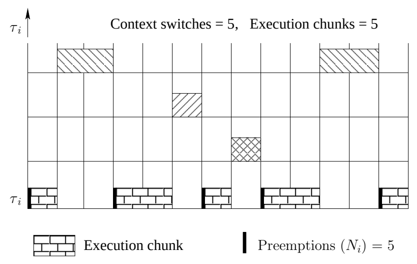


Fig. 7. Preemption count terminology

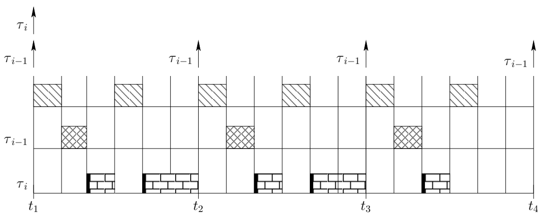


Fig. 8. Overlapping windows of τ_{i-1}

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 3). Therefore every job of a process is preempted an identical number of times. For this case, we now develop a technique to compute the number of preemptions.

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 equation gives an upper bound for N_i.

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

In this equation we assume $\Theta_i^{(0)} = \Theta_i$ and $\Theta_i^{(k)} = \Theta_i + N_i^{(k-1)} \delta_p$, where δ_p denotes the execution overhead for each preemption. 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 $N_i^* \geq N_i$.

Proof: 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. For example, in Figure 8 there are three dispatches of τ_{i-1} that overlap with the execution of τ_i: intervals $(t_1, t_2], (t_2, t_3]$ and $(t_3, t_4]$. We then determine the number of preemptions incurred by τ_i within the execution window of each these dispatches of τ_{i-1}. Since every job of a process executes in the same time instants relative to its release time, this number of preemptions is the same in each of these execution windows, except the last one. In the last window it is smaller because the execution of τ_i can terminate before the end of the window. Use of ceiling function in the equation implies that the last window is treated similar to the other execution windows, and this is one factor for the upper bound.

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 [6]. The maximum possible number of chunks is given by Π_{i-1} / Π_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. For each higher priority process τ_j, Π_{i-1} / Π_j gives the number of jobs of τ_j in the current execution window and N_j gives the number of preemptions incurred by each of those jobs. Then the number of execution chunks of τ_j in the entire window is $(N_j \times \Pi_{i-1}) / \Pi_j$. However all of these execution chunks cannot be always discarded; specifically the last one. Since the response time of τ_j need not necessarily coincide with a release of τ_1, τ_i could potentially continue its execution immediately after the last execution chunk of τ_j. In Equation (11) we always use $N_j - 1$ for the number of execution chunks of τ_j and this results in an upper bound. ■

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 exact preemption counts for processes $\tau_1, \ldots, \tau_{i-1}$. We now present a technique that overcomes these shortcomings. We modify Equation (11) as follows:

- We replace the ceiling function with a 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 or not.

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

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

[6]Note that the number of execution chunks is always equal to the number of preemptions encountered by the process.

We now give equations to compute the two unknown quantities I_j and $N_i^{(k)'}$ in the above formula.

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

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} (for example, t_3 in Figure 8). $\mathrm{R}_i^{(k)} - T_{i-1}^{(k)}$ 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 $N_i^* = N_i$.

One may argue that the exact preemption count can also be obtained by simulating the execution of processes. Since process periods are harmonic, the simulation 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 $\tau_i = (0, 0, \Pi_i, \Theta_i + N_i\delta_p, \Pi_i)$. If the resulting process set $\{\tau_1, \ldots, \tau_n\}$ is schedulable[7], 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 the principled design of partition level schedules. They also provide analytical correctness guarantees, which can be used in system certification.

References

[1] *ARINC specification 653-2, part I*, Engineering Standards for Avionics and Cabin Systems (AEEC), 2006.

[2] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proceedings of Euromicro Conference on Real-Time Systems*, 2003, pp. 151–158.

[3] X. Feng and A. Mok, "A model of hierarchical real-time virtual resources," in *Proceedings of IEEE Real-Time Systems Symposium*, 2002, pp. 26–35.

[4] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of IEEE Real-Time Systems Symposium*, 2003, pp. 2–13.

[5] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Proceedings of IEEE Real-Time Systems Symposium*, 2007, pp. 129–138.

[6] K. Tindell, "Adding time-offsets to schedulability analysis," Department of Computer Science, University of York, Tech. Rep. YCS 221, 1994.

[7] M. D. Natale and J. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," in *Proceedings of IEEE Real-Time Systems Symposium*, 1994, pp. 216–227.

[8] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

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

[10] Green Hills Software Inc., "ARINC 653 partition scheduler," www.ghs.com/products/safety_critical/arinc653.html.

[11] Windriver Inc., "Platform for ARINC 653," www.windriver.com/products/platforms/safety_critical/.

[12] L. Kinnan, J. Wlad, and P. Rogers, "Porting applications to an ARINC 653 compliant IMA platform using Vxworks as an example," in *Proceedings of IEEE Digital Avionics Systems Conference*, 2004.

[13] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling tool and algorithm for integrated modular avionics systems," in *Proceedings of IEEE Digital Avionics Systems Conference*, 2000.

[14] A. Mok, D.-C. Tsou, and R. de Rooij, "The MSP.RTL real-time scheduler synthesis tool," in *Proceedings of IEEE Real-Time Systems Symposium*, 1996, pp. 118–128.

[15] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: Response-time analysis and server design," in *Proceedings of ACM & IEEE International Conference on Embedded Software*, 2004, pp. 95–103.

[16] R. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proceedings of IEEE Real-Time Systems Symposium*, 2005, pp. 389–398.

[17] S. Matic and T. Henzinger, "Trading end-to-end latency for composability," in *Proceedings of IEEE Real-Time Systems Symposium*, 2005, pp. 99–110.

[18] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *Proceedings of IEEE Real-Time Systems Symposium*, 2006, pp. 257–270.

[19] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems," in *Proceedings of ACM & IEEE International Conference on Embedded Software*, 2007, pp. 279–288.

[20] N. Fisher, M. Bertogna, and S. Baruah, "The design of an edf-scheduled resource-sharing open environment," in *Proceedings of IEEE Real-Time Systems Symposium*, 2007, pp. 83–92.

[21] 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 *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 1996, pp. 195–202.

[22] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, vol. 40, pp. 117–134, 1994.

[23] J. Goossens, "Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints," Ph.D. dissertation, Universit Libre de Bruxelles, 1999.

[24] A. Easwaran, I. Shin, I. Lee, and O. Sokolsky, "Bounding preemptions under EDF and RM schedulers," University of Pennsylvania, Tech. Rep. MS–CIS–06–06, 2006.

[25] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[7] Liu and Layland have given response time based schedulability conditions for this case [25].