Schedulability Bound for Integrated Modular Avionics Partitions

Jung-Eun Kim, Tarek Abdelzaher and Lui Sha
University of Illinois at Urbana-Champaign, Urbana, IL 61802
Email:{jekim314, zaher, lrs}@illinois.edu

***Abstract*—In the avionics industry, as a hierarchical scheduling architecture Integrated Modular Avionics System has been widely adopted for its isolating capability. In practice, in an early development phase, a system developer does not know much about task execution times, but only task periods and IMA partition information. In such a case the schedulability bound for a task in a given partition tells a developer how much of the execution time the task can have to be schedulable. Once the developer knows the bound, then the developer can deal with any combination of execution times under the bound, which is safe in terms of schedulability. We formulate the problem as linear programming that is commonly used in the avionics industry for schedulability analysis, and compare the bound with other existing ones which are obtained with no period information.**

I. Introduction

The avionics industry has widely adopted the Integrated Modular Avionics (IMA) architecture [1, 23], which enables real-time functions to run within partitions that are temporally and spatially isolated from one another. Real-time tasks run within an IMA partition which is an execution environment of software applications according to the ARINC 653 standard [2]. Since IMA supports the partitions' temporal and spatial isolation from one another, the various real-time avionics functions can be developed independently. In addition to that, the mechanism has helped ease the certification process for mixed-criticality avionics systems. If done correctly, this modular approach can avoid substantial re-certification costs.

In practice, in an early development phase of IMA systems, IMA partition parameters and task periods are predetermined while the worst-case execution times of the tasks are partially or even completely unavailable. Such a situation is common. For example, when receiving data from a sensor, a task may know the sensing frequency in advance but not the worst-case execution time since determining the worst-case execution time is not a simple job. In such a case, if we could know the schedulability bound with no information on task execution time, that would be helpful to determine whether certain tasks are able to be integrated into a system or not. Besides, in the course of system development, if a system designer would add/modify a task to/on an existing system, the designer needs to check the schedulability of the system including the new/modified task. If it is schedulable the new/modified task can be included into the system. In such cases the schedulability test provides a quick, abstracted assessment of the system under development.

Hence, in this paper, we present an analysis determining the schedulability bound for a set of tasks in an IMA partition, when IMA partition information and *task periods are known while task execution times are unavailable*. With the bound, we can determine *how much of execution times the tasks can have to be schedulable in the given periods* in a hierarchical IMA system. Since we can determine the bound when there is no information about task execution times at all, we can also trivially do it when only a subset of information for task execution times is available.

To derive the bound, we formulate the problem as a linear programming (LP) problem that can be solved by a linear programming solver which is commonly used in the avionics industry for schedulability analysis. If the final bound is above or equal to the total utilization of all tasks in the partition, the tasks are determined to be schedulable. Throughout this paper, we assume task execution on an IMA system is based on Rate-Monotonic (RM) scheduling [20] which is the optimal scheduling policy on fixed priority scheduling and supported by open standards in avionics systems. RM assigns higher priority on the task with a shorter period.

In order to see the impact of more known information, we conduct comparisons in Sec. IV. There are two works which look for a utilization bound in a hierarchical environment with resource information in the upper level but without task information. In [25], a scheduling bound is presented when only Real-Time Virtual Machine (RTVM) sizes are given while task information is not. RTVM can be seen as a similar hierarchical concept with IMA system in this paper. The bound shows the maximally possible utilization of tasks in an RTVM. In [27], considering a periodic resource, a utilization bound for RM is presented when only resource information but no task information (workload) is given (only the number of tasks, the shortest task period and the ratio between periods are known). The resource is similar to IMA partition in the perspective of a task in our context. Since the utilization bound can show the maximum load of tasks under a certain resource requirement, it can be used for determining whether to accept a (new) task into a system or not (admission test). Since our presented bound is a result based on 'more' information (task periods) than that of [25] or [27], our bound is supposed to be higher than theirs.

On the stream of the research of temporally partitioned hierarchical scheduling there have been two main issues - one is how much of task utilization can be supported by the allocated resource requirement. That is, as what this paper is about, to find the maximal utilization of tasks which run under a given partition (e.g., server, resource, RTVM) requirement. Besides to ours, [25] and [27] reside in this category. The another issue on the stream is, the other way around, how much of the computational resource needs to be allocated to each partition in order for the system to be optimized for a certain metric. For instance, it is desirable in the system design process to minimize the system utilization while guaranteeing the timing requirements of both partitions (servers) and their applications. That is obtaining the partition or server parameters to sustain the given task. Relevant work can be found in [3, 5–8, 18, 19, 24, 27, 29, 30].

Our LP formulation is based on the one in [16] which used

978-3-9815370-4-8/DATE15/©2015 EDAA

the formulation for QoS management in admission control, thus the formulation is partly found in [16]. However, the formulation works on only a *non-hierarchical* environment, while in this paper we formulate the LP to deal with hierarchical IMA structure. The authors [21] presented an LP formulation which is not polynomial for an analogous problem with one in [16]. In non-hierarchical system for this problem which regards no period information, the authors of [13, 14] showed the utilization bound depending on the number of harmonic groups of tasks. On the other hand the authors in [15] presented another bound regarding the ratio between the longest and shortest period. An exponential complexity algorithm in the integer domain for that problem was presented in [4]. More other relevant work can be found in [9, 12, 17].

II. System Model

A. Integrated Modular Avionics System

In IMA system, a partition is a hierarchical running environment for tasks. Application tasks belong to an IMA partition and run only when their partition is active, which achieves temporal isolation between partitions. Once their assigned partition finishes, any running task should be suspended. Since partitions are based on cyclic executive, a partition is assigned by certain time slots to run, and the assigned schedule repeats every *major cycle*. For example in Fig. 1 (a), slot 3 is assigned to partition 2 and slots 5–7 are assigned for partition 3 while slot 2 and 4 are not assigned to any partition, and this partition schedule repeats every 10.

Tasks in different partitions do not share the same address space or any global variables, and an IMA system does not allow dynamic resource allocation such as dynamic memory allocation. These have partitions be spatially isolated, and prevent any failure propagation from a partition to another, which is the aim of designing IMA system [23]. Also, device I/O transaction atomically begins and finishes within a single slot.[1] In addition to that, each IMA partition can have its own scheduling policy, which eases partition-level development, migration or certification. Throughout this paper, we assume that tasks run on RM scheduling policy.

B. System Description

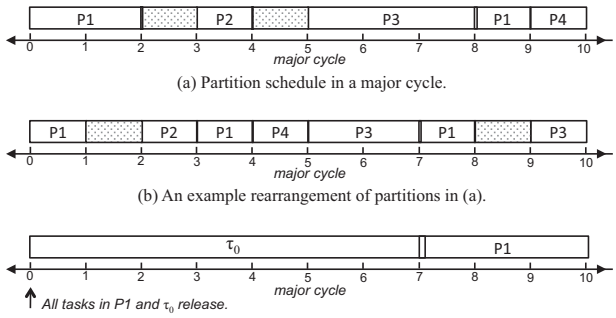
At the higher level in a hierarchical IMA system, a partition contains a set of periodic application tasks $\mathbf{\Gamma} = \{\tau_i | i = 1, \ldots, n\}$. Each τ_i is represented by $\tau_i := (e_i, p_i)$ where e_i is the worst-case execution time and p_i is the period which is equal to the relative deadline. Without loss of generality, tasks are sorted according to their priorities in decreasing order, i.e., τ_1 is the highest priority and τ_n is the lowest, that is, p_1 is the shortest and p_n is the longest according to RM policy. We assume no task release jitter. Context switching overheads of application tasks and partitions are assumed to be zero.

The total utilization of all the application tasks in a single partition is denoted by U_{total} and defined as follows:

Definition 1. $U_{total} = \sum_{i=1}^{n} \frac{e_i}{p_i}$.

Also, IMA *partition capacity* is defined as follows:

[1] Actually, an I/O transaction is done in a special-purpose partition called zero partition or device management partition to perform I/O transactions in a consolidated fashion and thus has simplified both implementation and management of I/O operations. However, it is out of scope of this paper. Interested readers can refer to [10, 11, 22, 23] for the full details.



(a) Partition schedule in a major cycle.

(b) An example rearrangement of partitions in (a).

(c) Overall system schedule in the perspective of a task in P1 when all tasks and τ_0 release at time t=0, i.e., in the worst-case. Note that τ_0 has the highest priority.

Fig. 1: Partition schedule.

Definition 2. *Partition capacity* $= \frac{\textit{\# of assigned slots}}{\textit{major cycle}}$.

For instance, in Fig. 1 (a) and (b), partition 1's capacity is $\frac{3}{10} = 0.3$ and partition 2's is $\frac{1}{10} = 0.1$. In our model, the major cycle is not longer than any task's period. However, this *never* implies that if the major cycle is longer than a task's period the task misses its deadline. For example, if we are given task $\tau_i = (e_i, p_i) = (1, 5)$, (deadline$= 5$) and a partition with major cycle of 10, the task always meets its deadline wherever the partition slots are allocated as long as the partition takes 6 or more slots (*i.e.*, partition's capacity ≥ 0.6).

This issue is also connected with the worst-case partition slot arrangement. A task in a partition experiences the worst-case response time when it releases at (*i.e.*, just right after) the finishing instant of its partition and all the partition slots aggregate together (see [6, 25]). This is visually described in Fig. 1 (c). Partition 1's slots are consecutively allocated from $t = 7$ to 10, and a task in partition 1 releases at $t = 0$ ($= 10, \because$ the major cycle is 10), which is the worst-case situation for the task regardless of other partitions' arrangement, since the task's slot allowed to run was just past, thus it should wait to run until the next cycle.

In order to model the worst-case situation, we apply the concept of VM Periodic Task in [25] to a single task, τ_0. τ_0 is relatively defined for the tasks in each partition.

Definition 3. *τ_0: in the perspective of a task in a partition,*

- *τ_0 is the highest priority task in the entire system, and*
- *τ_0's execution time is the sum of the slots not assigned to its own partition, and*
- *τ_0's period is the major cycle.*

For example, whatever the original schedule was one such as Fig. 1 (a) or (b), in the perspective a task in partition 1, τ_0's execution time $e_0 = 7$ and period $p_0 = 10 =$ major cycle, which can be represented as Fig. 1 (c). Since τ_0 holds the highest priority, the task experiences the worst-case response time when it simultaneously releases with τ_0 at $t = 0$ by the critical instant theorem of Liu & Layland [20]. In a hierarchical partition scheduling, a task cannot run on the unassigned slots (other partition's assigned slots and the empty slots) but can run only on its own partition's assigned slots. By keeping the highest priority and having execution time as much as the sum of unassigned slots for the partition, τ_0 realizes the effect. Since τ_0 inherently models the worst-case effect in the schedulability of a task, now an arrangement of partitions does not affect the schedulability and thus can be flexible. That is, the partition schedule in Fig. 1 (a) can be converted into the

one in Fig. 1 (b) or anything as long as all the partitions' capacities are still the same, which means that we only need partition capacities but not an actual arrangement.

C. Problem Description

The problem is stated as follows: for a partition, we derive the maximal sufficient bound, $U_{bound}(\tau_i)$ for task τ_i, that minimizes the total tasks' utilization when

- major cycle and partition capacity (*i.e.*, $\tau_0 = (e_0, p_0)$), and
- application tasks' periods, i.e., $\{p_k | k = 1, \ldots, i\}$

are given. Then the maximal sufficient bound for all the tasks is denoted by U_{bound} and represented by

$$U_{bound} = \min_{1 \le i \le n} \Big(U_{bound}(\tau_i) \Big).$$

III. Maximal Sufficient Utilization Bound

We start our analysis by deriving the schedulability bound of each task in a partition, $U_{bound}(\tau_i)$. Then the minimum among each resulted bound is *the* bound, U_{bound}. If $U_{bound} \ge U_{total}$, the set of tasks in the partition is schedulable, otherwise it is not. Hence, we start deriving the schedulability bound for each task τ_i $(1 \le i \le n)$ in a partition.

A. Bound for Each Task in a Partition, $U_{bound}(\tau_i)$

A trivial sufficient bound to guarantee schedulability is an arbitrarily small value which is not useful in practice. Thus, we need to obtain the maximal sufficient bound for task τ_i. To find the maximal sufficient condition for task τ_i, we search for a task set with the minimal utilization in a partition characterized by the following three conditions identified by Liu and Layland in [20]. We shall refer them as ***L&L conditions***.

- **Condition 1:** The preemption to task τ_i is maximized by having all the tasks simultaneously begin at time $t = 0$.
- **Condition 2:** Task τ_i completes before its first deadline.
- **Condition 3:** The processor is fully utilized in $[0, p_i]$.

Since task periods are given, with those constant periods the maximal sufficient bound can be obtained by a linear programming as being subject to L&L conditions.

Condition 1 is easily addressed by indexing all the tasks' execution start times at $t = 0$. Condition 2 is also easy to be represented, which can be checked by summing up all the preemptions on task τ_i and τ_i's execution time, and then ensuring if the sum is larger than τ_i's deadline $(= p_i)$ or not.

However, Condition 3 cannot be easily addressed since it would need a potentially large number of constraints. Since Condition 3 ensures that the bound is *the maximal* sufficient bound not just a sufficient one, it requires the time duration $[0, p_i]$ to be fully utilized. To satisfy the condition, no idle time is supposed to be in $[0, p_i]$. If there is an idle time in $[0, p_i]$ the bound would not be maximally sufficient, since that means there is still room to be utilized as much as the idle time. Thus, execution times can increase until the duration $[0, p_i]$ gets filled – we call this process as a *maximalization* process which leads a sufficient bound to the maximally sufficient one.

1) Linear Programming Formulation: For the first step to represent Condition 3, let us account for the preemption in a form of linear constraints. In Fig. 2, we are interested in the schedulability of τ_3. From the perspective of task τ_3, the preemption from each higher priority task can be divided into two portions - *overflow* part and *non-overflow* part. Since

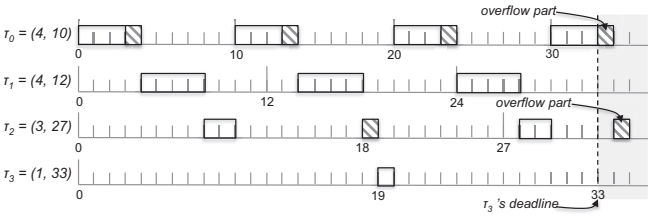


Fig. 2: Preemption on τ_3, by non-overflow and overflow part.

the deadline of τ_3 is 33, the execution of the higher priority tasks of τ_3 which occurs after 33 is classified as overflow part, while the remaining execution which happens before 33 is non-overflow part. This classification applies to any instance. For example, since one slot of τ_0's forth instance runs after τ_3's deadline 33, in every τ_0's instance overflow part for τ_3 denoted by $e^{over}_{0\to3}$ is 1 (diagonally striped in Fig. 2), while the non-overflow part denoted by $e^{non}_{0\to3}$ is 3. Likewise, τ_1's overflow part, $e^{over}_{1\to3} = 0$ and τ_2's $e^{over}_{2\to3} = 1$. For a bound of task τ_i, generally, for its higher task τ_h, $e^{over}_{h\to i} = \max\Big(\Big(\Big\lfloor \frac{p_i}{p_h} \Big\rfloor \cdot p_h + e_h\Big) - p_i, 0\Big)$, and $e^{non}_{h\to i} = e_h - e^{over}_{h\to i}$.

In the example, the non-overflow part of τ_0 preempts τ_3 four times, while its overflow part preempts τ_3 only three times. Generally, the total preemption from a high priority task τ_h to τ_i is $\Big\lceil \frac{p_i}{p_h} \Big\rceil e^{non}_{h\to i} + \Big\lfloor \frac{p_i}{p_h} \Big\rfloor e^{over}_{h\to i}$. Hence, the total preemption to task τ_i from all the higher priority tasks can be represented as $\sum_{h=0}^{i-1} \Big(\Big\lceil \frac{p_i}{p_h} \Big\rceil e^{non}_{h\to i} + \Big\lfloor \frac{p_i}{p_h} \Big\rfloor e^{over}_{h\to i} \Big)$. Now, suppose that there exists a set of linear constraints $\mathbf{R}^*$ (will be shown later), in which $e^{non}_{h\to i} (0 \le h \le i\text{–}1)$ still stay as non-overflow parts during a maximalization process and the processor is fully utilized in the interval $[0, p_i]$. Now let us define $U_{bound}(\tau_i)$.

Definition 4. $\mathbf{U_{bound}}(\tau_\mathbf{i})$*: denotes the maximal sufficient bound for task* τ_i *in the set of tasks* $\{\tau_k | 1 \le k \le i\}$.

Lemma 1. *The maximal sufficient bound* $U_{bound}(\tau_i)$ *for scheduling* τ_i *is obtained by the linear programming problem,*

$$U_{bound}(\tau_i) = \min \Big(\sum_{k=1}^{i} \frac{e_k}{p_k} \Big)$$

subject to $\mathbf{R}^*$

and
$$\sum_{h=0}^{i-1} \Big(\Big\lceil \frac{p_i}{p_h} \Big\rceil e^{non}_{h\to i} + \Big\lfloor \frac{p_i}{p_h} \Big\rfloor e^{over}_{h\to i} \Big) + e_i = p_i. \quad (1)$$

where $e^{non}_{0\to i}, e^{over}_{0\to i}$, *and* $p_k (0 \le k \le i)$ *are constant*; $e_k (1 \le k \le i)$ *are decision variables*.

Proof. It directly follows L&L conditions - (1) represents Condition 1 and 2, and $\mathbf{R}^*$ does Condition 3. □

Before identifying $\mathbf{R}^*$, we will eliminate all overflow variables in (1) by setting the values to zero, except that of τ_0. Since for every task p_i is constant, the overflow part of e_0 does not change. Thus, after maximalization process, we would not need to explicitly list the overflow variables (except τ_0's).

Lemma 2. *When the solution of the linear programming problem in Lemma 1 is attained,* $e^{over}_{h\to i} = 0 \quad (1 \le h \le i-1)$.

Proof. (Although similar proving can be found in Lemma 1 in [16], we prove it differently here for our context.) Assume that the minimal utilization task set has at least one overflow variable which is greater than 0, *i.e.*, $e^{over}_{h\to i} > 0$, for some h. Then the processor utilization contributed by $e^{over}_{h\to i}$ is $\frac{e^{over}_{h\to i}}{p_h}$

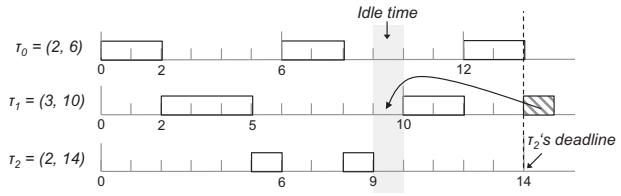


Fig. 3: Interval $[0, 14]$ is not fully utilized (there is idle time) in spite of satisfying the equality constraint (2).

and the preemption provided by $e^{over}_{h\to i}$ is $\left\lfloor \frac{p_i}{p_h} \right\rfloor e^{over}_{h\to i}$. We keep the processor busy during $[0, p_i]$ by setting $e^{over}_{h\to i} = 0$ and increasing e_i by $\left\lfloor \frac{p_i}{p_h} \right\rfloor e^{over}_{h\to i}$. Then the reduced utilization is $\frac{e^{over}_{h\to i}}{p_h}$ and the increased utilization is $(\left\lfloor \frac{p_i}{p_h} \right\rfloor e^{over}_{h\to i})/p_i$. Accordingly, the net processor utilization is reduced since

$$\frac{p_i}{p_h} e^{over}_{h\to i} \geq \left\lfloor \frac{p_i}{p_h} \right\rfloor e^{over}_{h\to i},$$

$$\frac{e^{over}_{h\to i}}{p_h} \geq \frac{\left\lfloor \frac{p_i}{p_h} \right\rfloor e^{over}_{h\to i}}{p_i},$$

reduced utilization $\geq$ increased utilization.

Hence, the net is reduced while $[0, p_i]$ is fully utilized. This contradicts the assumption that the minimal utilization task set has a non-zero overflow variable except for τ_0. □

According to Lemma 2, now we have $e^{over}_{h\to i} = 0$ $(1 \leq h \leq i-1)$, and thus in Lemma 1 we can eliminate overflow variables, $e^{over}_{h\to i} (1 \leq h \leq i-1)$. Applying Lemma 2, the rewritten form of Lemma 1 is in Lemma 3 as follows:

Lemma 3. *The maximal sufficient bound $U_{bound}(\tau_i)$ for scheduling τ_i is obtained by the linear programming problem,*

$$U_{bound}(\tau_i) = \min \Big(\sum_{k=1}^{i} \frac{e_k}{p_k} \Big)$$

subject to $\mathbf{R}^*$

$$\textit{and } \Big(\Big\lceil \frac{p_i}{p_0} \Big\rceil e^{non}_{0\to i} + \Big\lfloor \frac{p_i}{p_0} \Big\rfloor e^{over}_{0\to i} \Big) + \sum_{h=1}^{i-1} \Big(\Big\lceil \frac{p_i}{p_h} \Big\rceil e_h \Big) + e_i = p_i \quad (2)$$

where

$e^{non}_{0\to i}, e^{over}_{0\to i}$, *and* $p_k (0 \leq k \leq i)$ *are constant*; $e_0 = e^{non}_{0\to i} + e^{over}_{0\to i}$;

$z_k (1 \leq k \leq \sum_{h=0}^{i-1} \lceil \frac{p_i}{p_h} \rceil)$: *series of all arrival instants in* $(0, p_i)$;

$e_k (1 \leq k \leq i)$ *are decision variables*.

Proof. It directly follows Lemma 1 and Lemma 2. □

The variables for arrival instants, z_k, are needed for $\mathbf{R}^*$, of which idea is also found in the formulation of Theorem 2 in [16]. Let us first identify $\mathbf{R}^*$ which ensures that the processor is busy in $[0, p_i]$ and the non-overflow variables are indeed non-overflow parts during the maximalization process. To do so, we first remove the $\mathbf{R}^*$ constraint from Lemma 3 and observe the implication. In Fig. 3. Consider the case of three tasks, $\tau_0(2, 6), \tau_1(3, 10)$ and $\tau_2(2, 14)$. These three tasks satisfy the equality constraint (2) in Lemma 3 as follows:

$$\Big(\Big\lceil \frac{p_2}{p_0} \Big\rceil e^{non}_{0\to 2} + \Big\lfloor \frac{p_2}{p_0} \Big\rfloor e^{over}_{0\to 2} \Big) + \Big(\Big\lceil \frac{p_2}{p_1} \Big\rceil e_1 \Big) + e_2 = p_2,$$

$$\Big(\Big\lceil \frac{14}{6} \Big\rceil 2 + \Big\lfloor \frac{14}{6} \Big\rfloor 0 \Big) + \Big(\Big\lceil \frac{14}{10} \Big\rceil 3 \Big) + 2 = 6 + 6 + 2 = 14.$$

However, as we can see there is an idle time in [9,10] since the 2^{nd} invocation of τ_1 overflows. This cannot be checked only by the constraint (2) since it just ensures that the aggregate sum of all executions is p_i=14. Hence, we need Lemma 4 for non-overflow variables not to represent overflow executions.

Lemma 4. *By equality constraint* (2) *in Lemma 3, overflow in a high priority task's execution time implies that there exists an idle time in the interval* $[0, p_i]$.

Proof. Since the total processor time requested by all the tasks is equal to p_i, if any part of a high priority task executes after p_i, there must be an idle interval in $[0, p_i]$. □

Corollary 1. *By equality constraint* (2) *in Lemma 3, having no idle time in interval* $[0, p_i]$ *implies that all decision variables in Lemma 3 have no overflow parts, i.e.,* $e^{over}_{h\to i} = 0$ $(1 \leq h \leq i-1)$.

Proof. This is the contrapositive of Lemma 4. Since Lemma 4 is true, this proposition is true, as well. □

Therefore, constraint $\mathbf{R}^*$ can be represented by a set of linear constraints which forces all possible gaps, *i.e.*, idle time intervals, to be filled with executions. In Fig. 3, the task arrival instants (except $t = 0$) are $z_1 = 6, z_2 = 10$ and $z_3 = 12$. Note that once an idle interval starts (if any), it must terminate at the right next task arrival instant anyway. Because at that point, the next new task arrives. In the example, the idle interval beginning at $t = 9$ terminates at $t = 10$. Hence, the constraint ensuring that the total processor demand at an arrival instant is greater than or equal to (the instant -0), should be checked at every arrival instant. That is represented at (4) in Theorem 1.

Theorem 1. *The maximal sufficient bound $U_{bound}(\tau_i)$ for scheduling τ_i is obtained by the linear programming problem,*

$$U_{bound}(\tau_i) = \min \Big(\sum_{k=1}^{i} \frac{e_k}{p_k} \Big)$$

subject to

$$\Big(\Big\lceil \frac{p_i}{p_0} \Big\rceil e^{non}_{0\to i} + \Big\lfloor \frac{p_i}{p_0} \Big\rfloor e^{over}_{0\to i} \Big) + \sum_{h=1}^{i-1} \Big(\Big\lceil \frac{p_i}{p_h} \Big\rceil e_h \Big) + e_i = p_i \quad (3)$$

$$\textit{and} \quad \sum_{h=0}^{i-1} \Big(\Big\lceil \frac{z_k}{p_h} \Big\rceil e_h \Big) + e_i \geq z_k \quad (1 \leq k \leq \sum_{h=0}^{i-1} \Big\lfloor \frac{p_i}{p_h} \Big\rfloor) \quad (4)$$

where

$e^{non}_{0\to i}, e^{over}_{0\to i}$, *and* $p_k (0 \leq k \leq i)$ *are constant*; $e_0 = e^{non}_{0\to i} + e^{over}_{0\to i}$;

$z_k (1 \leq k \leq \sum_{h=0}^{i-1} \lceil \frac{p_i}{p_h} \rceil)$: *series of all arrival instants in* $(0, p_i)$;

$e_k (1 \leq k \leq i)$ *are decision variables*.

Proof. The proving follows a part of proof of Theorem 2 in [16]. If there is an idle interval in $[0, p_i]$, it will be terminated by either an arrival of a higher priority task before $t = p_i$ or at $t = p_i$. However, either case does not happen since it contradicts the linear programming constraints that there cannot be an idle time before each task arrives. By Corollary 1, disappearing of idle time makes an overflow impossible. Finally, the equality constraint (3) ensures that task τ_i is schedulable in $[0, p_i]$. In summary, Theorem 1's linear constraints satisfy all the L&L conditions. Furthermore, there cannot be any overflow (except for τ_0), and thus we do not need overflow variables. It follows that $U_{bound}(\tau_i)$ is the maximal sufficient bound. □

B. Bound for All Tasks in a Partition, U_{bound}

Now we derive the maximal sufficient bound, U_{bound}, for all the tasks $\{\tau_i | i = 1, \ldots, n\}$ in a partition, by using the bound

TABLE I: Utilization bounds according to partition capacity for Example 1. No information on task is given for SHA's and Shin's while we are given task periods. The more known information leads to a higher bounds. ($U_{bound} = min(U_{bound}(\tau_1),\ U_{bound}(\tau_2))$)

partition capacity	$U_{bound}(\tau_1)$	$U_{bound}(\tau_2)$	U_{bound}	SHA's	Shin's
0.1	0.08	0.08	0.08	0.05	minus
0.3	0.25	0.25	0.25	0.16	0.001
0.5	0.41	0.43	0.41	0.28	0.12
0.7	0.58	0.62	0.58	0.43	0.33
0.9	0.83	0.82	0.82	0.59	0.64

for each task, $U_{bound}(\tau_i)$, that we obtain in Section III-A.

Theorem 2. *A set of tasks $\{\tau_i | i = 1, \ldots, n\}$ is schedulable if its total utilization U_{total} is less than or equal to U_{bound}, i.e., $U_{total} \leq U_{bound}$, where $U_{bound} = \min_{1 \leq i \leq n}(U_{bound}(\tau_i))$.*

Proof. According to that $U_{total} \leq U_{bound}$ and $U_{bound} = \min_{1 \leq i \leq n}(U_{bound}(\tau_i))$,

$$U_{total} \leq U_{bound} = \min_{1 \leq i \leq n}(U_{bound}(\tau_i)). \quad (5)$$

$$\text{For } 1 \leq i \leq n, \quad \sum_{k=1}^{i} \frac{e_k}{p_k} \leq \sum_{k=1}^{n} \frac{e_k}{p_k} = U_{total}, \quad \text{and} \quad (6)$$

$$\min_{1 \leq i \leq n} \left(U_{bound}(\tau_i) \right) \leq U_{bound}(\tau_i). \quad (7)$$

According to (5), (6) and (7) for $1 \leq i \leq n$,

$$\sum_{k=1}^{i} \frac{e_k}{p_k} \leq \sum_{k=1}^{n} \frac{e_k}{p_k} = U_{total} \leq U_{bound} = \min_{1 \leq i \leq n} U_{bound}(\tau_i) \leq U_{bound}(\tau_i).$$

Finally, we can obtain $\sum_{k=1}^{i} \frac{e_k}{p_k} \leq U_{bound}(\tau_i) \quad (1 \leq i \leq n)$,

which shows that each task's scheduling bound is met. Thus, Theorem 2 holds. □

According to Theorem 2, ultimately the maximal sufficient utilization bound for a schedulable set of tasks in a partition, U_{bound}, is obtained by picking the minimum among $U_{bound}(\tau_1)$, $U_{bound}(\tau_2)$, $\ldots$, $U_{bound}(\tau_n)$. Let us see how the presented method works with a numerical example.

Example 1. *Suppose a partition and the major cycle is 10. In the partition there are two tasks, τ_1 and τ_2: τ_1's period p_1 is 12 and τ_2's period p_2 is 41.*

The resulted bounds according to partition capacity are shown in Table I. According to Theorem 1 and 2, U_{bound} is derived by picking the minimum among τ_1's and τ_2's bound, $U_{bound} = \min(U_{bound}(\tau_1), U_{bound}(\tau_2))$. For instance, for capacity 0.9, *i.e.*, $\tau_0 = (e_0, p_0) = (1, 10)$, $U_{bound} = \min(U_{bound}(\tau_1), U_{bound}(\tau_2)) = \min(0.83, 0.82) = 0.82$. Table I also shows the bound from [25] (referred to as SHA's) and [27] (referred to as Shin's).[2] More discussion continues in the next section.

IV. Evaluation

In Fig. 4 we compared the resulted bound from our LP with SHA's and Shin's for 1,000 samples: each sample task set contains 3–100 tasks with periods randomly generated from uniform distribution over 50–99 and major cycle over 30–60. As we can see the ratio between any two periods is less than 2 since Shin's bound (formulation (33) in [28]) can be applied

[2]For Shin's, the bound and theorem presented in [27] (full proof presented in [28]) contain errors thus the authors corrected them on [28] for their extended journal version [29]. Ultimately, bounds presented in Table I are obtained from (32) and (33) presented in the updated version of [28].

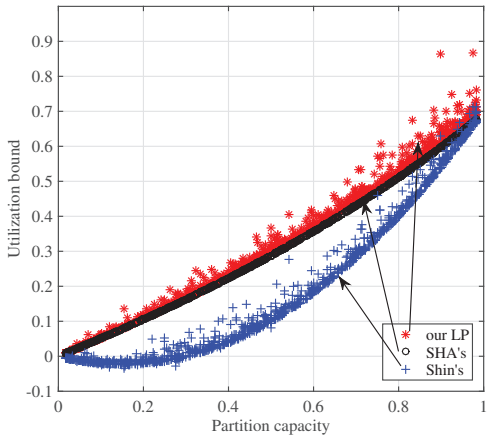


Fig. 4: (Shown in color) Comparison of utilization bounds only for task sets in which the ratio between any two periods is less than 2. Thus, the task sets are *favorable* for SHA's and Shin's.

only for such cases.[3] For that reason, our resulted bound is not that much higher than SHA's. Because in SHA's such task sets are assumed and then the bound is derived, which is favorable for SHA's bound. Hence, for a fair comparison, we ran another experiment with general periods for our LP and SHA's.

Fig. 5 shows the difference in utilization bound between our LP and SHA's according to a partition capacity. Differently from Fig. 4, in Fig. 5 ratio of a task's period to another can reach 30. Each task set also contains 3–100 tasks with periods randomly generated from uniform distribution over 10–300 and major cycle over 10–100. We ran 1,000 sample sets. The result shows that our LP enhances SHA's utilization bound since SHA's needs to suppose the worst-case periods and thus tries to reserve enough room for those tasks. Accordingly, the bound gets lower, *i.e.*, more conservative. On the other hand, since we are given the information for the task periods, we do not need to make the worst-case assumption on the task periods, which enables to spare more utilization for the tasks

[3]In [28], formulation (32) can be applied to a general case of periods, however, it can accommodate only 2 tasks.

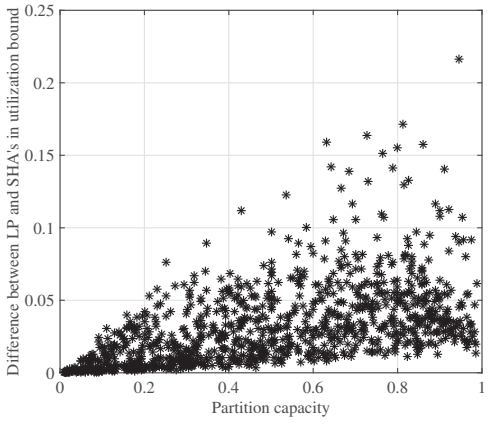


Fig. 5: The difference in utilization bound between LP solution and SHA's, for general cases of task sets in which the ratio between any two periods is not restricted by 2. This shows how much 'known information (= known periods)' enhances the bound.

and thus raise the bound.

One of the issues we empirically found in using the bound is that large remainders between a task's period and its higher priority tasks' lower the bound. The smaller the difference, the likelier the task is to have a higher bound. For an example, just change the period of τ_2 from 41 to 60 in Example 1 to make the remainders zero, that is, $\{p_0 = 10, p_1 = 12, p_2 = 60\}$. In the case of the partition capacity for 0.9, *i.e.*, $\tau_0 = (1, 10)$, τ_1's bound is 0.83 while τ_2's bound is 0.9 which is the full utilization of the partition capacity. That is because, τ_2's period is divisible by τ_0 and τ_1. Such a case also corresponds to the known fact that in RM harmonic tasks achieve 100% of utilization. For the reason, if we were given a task set with large remainders, for an enhanced utilization bound we could consider the use of the period transformation method such as the one described in [26], which transforms a period into a smaller period which is harmonic in the task set, if it is possible to apply to the application.

V. Conclusions

In this paper we formulated LP for a maximal sufficient bound for tasks in a given IMA partition without any information on task execution times. The bound will be a quick and convenient index for a system developer in an early development phase for a schedulability test of the tasks in the given partition. Because, once a developer knows the bound, the developer can deal with any other combination of the tasks with lower utilization than the obtained bound. To achieve the goal, our formulation seeks the maximal sufficient bound for each task with only periods and major cycle information. Then, ultimately provides the maximal sufficient bound for the entire task set. We believe that this is a useful work which can be employed in the field instantly. In addition to that, to see the impact of more known information on schedulability, we showed our bound with other existing ones which obtain a bound without task period values in a hierarchical system.

VI. Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments on the paper. This work is supported in part by Navy N00014-12-1-0046 and by NSF CNS 13-02563, 13-20209 and 14-23334. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of sponsors.

References

[1] *ARINC Specification 651: Design Guidance for Integrated Modular Avionics*. ARINC report. Airlines Electronic Engineering Committee and Aeronautical Radio Inc, Nov. 1991.

[2] Avionics application software standard interface: Arinc specification 653p1-3. *Aeronautical Radio, Inc.*, 2010.

[3] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM intl' conference on Embedded software*, pages 95–103, 2004.

[4] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound revisited. *IEEE Trans. Comput.*, 52(3), Mar. 2003.

[5] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 389–398, 2005.

[6] R. I. Davis and A. Burns. An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In *Proc. of Real-Time and Network Systems*, 2008.

[7] F. Dewan and N. Fisher. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '10, pages 247–256, 2010.

[8] A. Easwaran. Compositional schedulability analysis supporting associativity, optimality, dependency and concurrency. *PhD thesis, Computer and Information Science, University of Pennsylvania*, 2007.

[9] C.-C. Han and H.-Y. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 36–45. IEEE Computer Society, 1997.

[10] J.-E. Kim, M.-K. Yoon, S. Im, R. Bradford, and L. Sha. Optimized scheduling of multi-ima partitions with exclusive region for synchronized real-time multi-core systems. In *Proc. of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 970–975, 2013.

[11] J. Krodel. Commercial off-the-shelf real-time operating system and architectural considerations. *Federal Aviation Administration*, Feb. 2004.

[12] T.-W. Kuo, L.-P. Chang, Y.-H. Liu, and K.-J. Lin. Efficient online schedulability tests for real-time systems. *IEEE Trans. Software Eng.*, 29(8):734–751, 2003.

[13] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 160–170. IEEE Computer Society, 1991.

[14] T.-W. Kuo and A. K. Mok. Incremental reconfiguration and load adjustment in adaptive real-time systems. *IEEE Trans. Comput.*, 46(12):1313–1324, Dec. 1997.

[15] S. Lauzac, R. G. Melhem, and D. Moss. An efficient rms admission control and its application to multiprocessor scheduling. In *IPPS/SPDP*, pages 511–518, July 2003.

[16] C.-G. Lee, L. Sha, and A. Peddi. Enhanced utilization bounds for qos management. *IEEE Trans. Comput.*, 53(2):187–200, Feb. 2004.

[17] J. Liebeherr, A. Burchard, Y. Oh, and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. Comput.*, 44(12):1429–1442, Dec. 1995.

[18] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 151–158, 2003.

[19] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *J. Embedded Comput.*, 1(2):257–269, Apr. 2005.

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

[21] D.-W. Park, S. Natarajan, A. Kanevsky, and M. J. Kim. A generalized utilization bound test for fixed-priority real-time scheduling. In *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, 1995.

[22] P. Parkinson and L. Kinnan. Safety-critical software development for integrated modular avionics. *White Paper, Wind River Systems*, 2007.

[23] J. Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. *NASA Langley Technical Report*, Mar. 1999.

[24] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 152–160, 2002.

[25] L. Sha. Real-time virtual machines for avionics software porting and development. In *Real-Time and Embedded Computing Systems and Applications, the 9th International Conference, RTCSA*, Feb. 2003.

[26] L. Sha and J. B. Goodenough. Real-time scheduling theory and ada. *IEEE Computer*, 23(4):53–62, April 1990.

[27] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of the 24th IEEE International Real-Time Systems Symposium*, pages 2–13, 2003.

[28] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. Tech. rep., Dep. of Computer & Information Science, University of Pennsylvania, 2003 (rev. 2010).

[29] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3):30:1–30:39, May 2008.

[30] M.-K. Yoon, J.-E. Kim, R. Bradford, and L. Sha. Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling. In *Proc. of the 16th ACM/IEEE Design, Automation, and Test in Europe*, pages 1313 – 1318, 2013.