Real-time hierarchical systems with arbitrary scheduling at global level

Ana Guasque, Patricia Balbastre and Alfons Crespo*

Institute of Control Systems and Industrial Computing (ai2), Technical University of Valencia, Valencia, ES, 46022.

Abstract

Partitioned architectures isolate software components into independent partitions whose execution will not interfere with other partitions, preserving temporal and spatial isolation. Hierarchical scheduling can effectively be used to schedule these systems. Schedulability analysis of hierarchical real-time systems is based on prior knowledge of the local and the global scheduling algorithms.

In a partitioned system with safety and security issues and certification assurance levels, global scheduling is usually generated using a static table. Therefore, each partition must allocate task jobs only in the temporal windows reserved for that partition. Even if the static table can come originally from a periodic server or other scheduling policy, the final plan may be modified due to changes in the system requirements. As a consequence, the CPU assignment to a partition does not have to correspond to any known policy. In this case, it is not possible to use existing scheduling analysis for hierarchical systems.

This paper studies a new scheduling problem: a hierarchical system in which global policy is not known but provided as a set of arbitrary time windows.

Keywords: Real-time systems, partitioned systems, embedded systems, real-time systems scheduling, real-time algorithms.

*This work has been funded by the Spanish government under grant TIN2014-56158-C4-1-P-AR and by the European Commission under FP7-ICT-2013.3.4 Programme with grant 610640

Email address: `anguaor@ai2.upv.es, patricia@ai2.upv.es, acrespo@ai2.upv.es` (Ana Guasque, Patricia Balbastre and Alfons Crespo)

1. Introduction

In many domains such as avionics, space or industrial control systems, hard real-time constraints, safety and security issues and certification assurance levels are commonly required. Integrated Modular Avionics (IMA) is an architectural proposal that emerged as a design concept to integrate several applications with different levels of criticality in a hardware platform. The IMA approach proposes to encapsulate functions into partitions configuring a partitioned system. Partitioned architectures isolate software components into independent partitions whose execution must not interfere with others, preserving temporal and spatial isolation. Several projects have been successfully developed using this approach in the avionic market.

In the last decade, the European space sector has adapted the initial IMA approach for the space requirements for the new generation of satellites [1]. The IMA-SP project focused on mono-processors [2]. The platform defines a virtualization layer (hypervisor) that permits execution of several partitions. Each partition can contain a guest operating system and the application software. The hypervisor is in charge of ensuring temporal and spatial isolation of partitions.

An IMA development process involves several roles like:

- System Architect (SA): The SA is responsible for defining the overall system requirements and system design, including optimal decomposition into hosted partitions jointly with the detailed resource allocation per partition.
- System Integrator (SI): The SI is responsible for verifying the feasibility of the system requirements defined by the SA, as well as responsible for the configuration and integration of all components.
- Application Suppliers (AS): An AS is responsible for developing an application according to the overall requirements from the SA and the SI. AS must verify compliance with the allocated budget and safety parameters. Assuming that each application is located in a partition and a partition can have only one application, an AS can also be called Partition Developer (PD).

There are other roles in the process but due to space restrictions we only detail those interesting for the purpose of this article. For a complete description of the main roles and responsibilities see ([3]).

A key element in the development process and the final execution is the configuration of the system defined by the SA, which includes the description of the components and resource allocation. This is identified as configuration data or configuration file. In order to preserve the confidentiality of the development process, configuration data is split and delivered to each PD with the required information for developing the application.

The SI is responsible for CPU allocation of temporal resources to applications while the PD manages the time budget assigned to its tasks by the SI. Based on the proposed software architecture in an IMA system where a hypervisor supports the execution of several temporal and spatial isolated partitions, the system can be modeled as a hierarchical real-time system in which tasks are allocated to partitions. The SI allocates

CPU in the global level, according to the scheduling algorithm of its choice, while the PD internally schedules tasks with its own scheduling algorithm and the assigned CPU budget. The SI is responsible for ensuring feasibility in the global level while PD ensures feasibility in the corresponding local level. Figure 1 shows the structure of a partitioned system. The scheduling plan in the global level schedules partitions according to an offline plan defined in the static configuration file of the system.

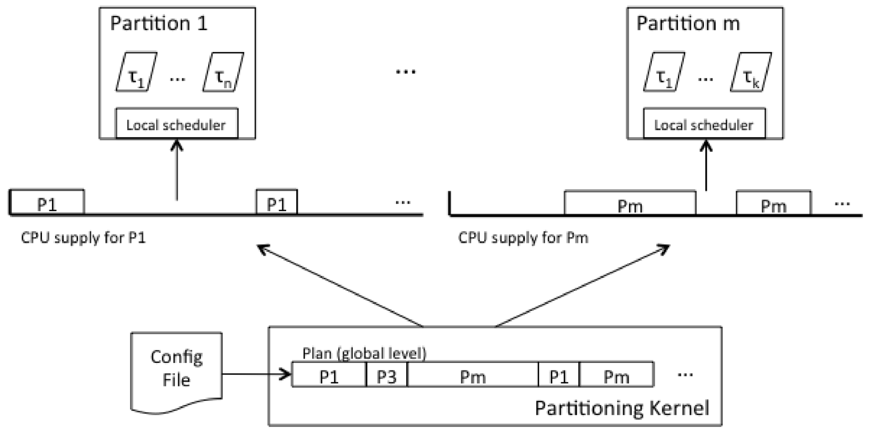


Figure 1: General overview of the partitioned system

Thus, a partition does not have all the time assigned to schedule its tasks, but only certain slots throughout the hyper-period. An example is shown in Figure 2, where a partition with a set of periodic tasks is scheduled under EDF (Earliest Deadline First) policy.

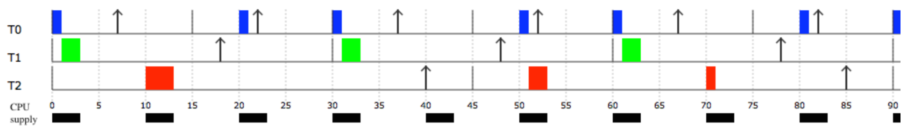


Figure 2: Execution chronogram and CPU supply of a partition

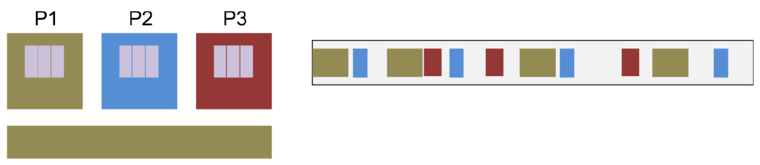
In the previous figure, the global scheduler peridically assigns the CPU to the task set, that is, the partition receives a periodic resource supply that provides 3 units of CPU every 10 units of time. The black rectangles at the bottom of the figure represent the slots assigned to the partition. Obviously, tasks cannot execute outside these slots, since they are reserved for other partitions.

The list of assigned slots is provided by the SI, responsible for ensuring the feasibility in the global level. Thus, PD gives the SI its temporal requirements, normally in the form of CPU bandwidth. The SI calculates and assigns this bandwidth to partitions using a well known bandwidth server or cyclic scheduling. ARINC 653 standard [4] defines a hierarchical scheduling where a static cyclic executive scheduler is used in the global level.

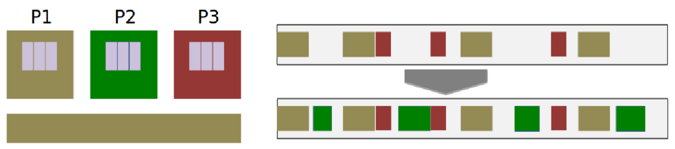
If the assignment is made using a bandwidth algorithm or a periodic resource model, the corresponding feasibility tests are available in the literature so the PD can apply them to know if its tasks are schedulable with this slots assignment (see section 8). On the contrary, if the SI makes the asignment arbitrarily (i.e. not following any existing scheduling algorithm) the authors are not aware of any article that addresses and solves this problem.

Below, we present an example of why a partition can be assigned an arbitrary sequence of slots. Let us assume a partitioned system with three partitions (P1, P2 and P3) and the scheduling plan shown in Figure 3(a).

If the temporal requirements of P2 changefor any reason, P2 will be scheduled in the empty slots not used by P1 and P3 (Figure 3(b)). These slots do not correspond to any periodic reservation so we can consider that the sequence of slots provided by SI to PD of P2 are arbitrary. Of course, we can also re-schedule the entire system but then the scheduling of P1 and P2 would change, requiring certification of partitions whose requirements do not change. Such an effort must be avoided if possible.



(a) Initial situation

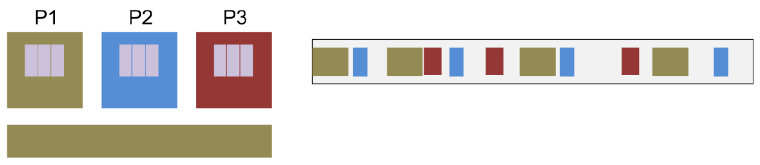


(b) One of the partitions changes its temporal parameters.

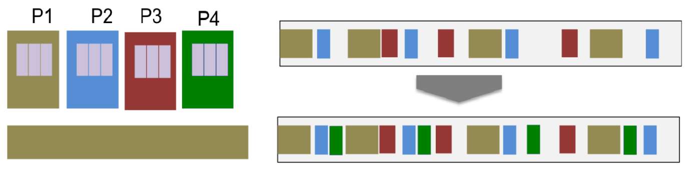
Figure 3: Changing temporal parameters in one partition.

If we later add a fourth partition to the system (P4) we will have to schedule P4 in the idle slots not used by P1, P2 and P3. Again, the slots reserved for P4 can be considered arbitrary (Figure 4(b)).

These two situations show two different scenarios where a partition must be scheduled in time slots that do not follow any known allocation. Schedulability tests for hierarchical systems are based on calculating the worst case response time of tasks in the local level and adding the worst case overhead due to the global level. This last overhead cannot be calculated if the scheduling policy in the global level is not known. Thus, the existing literature does not give a solution to this problem. For this reason, we provide a solution to analyze the schedulability of a task set of a partition where the scheduling algorithm is arbitrary at global level.



(a) Initial situation



(b) Addition of a new partition.

Figure 4: Adding partitions to the partitioned system.

1.1. Contributions and outline

The problem to be addressed is concerned with the schedulability of a hierarchical system composed of two levels. The global level policy is not known but provided by the SI as a set of arbitrary time slots. By arbitrary we understand that the sequence of slots is not derived from any known scheduling algorithm in the global level. This is the most important difference with respect to existing papers in the area.

This article provides a method for determining a sequence of slots provided by the SI that makes the local level schedulable. Specifically, we define two different slots assignments.

We also provide a basic schedulability analysis so the PD can accept the slots assignment. In the local level, we assume EDF. Obviously, our results can be used even if the scheduling algorithm in the global level is known.

The article is organized as follows: Section 2 presents the model and notation used, while in section 3 the calculation of schedulable supply bound functions is explained. Section 4 presents some results for the schedulability areas defined between the supply bound functions presented in the previous section. The schedulability test is presented in section 5. Section 6 contains the evaluation of our proposal while section 7 presents a comparison with a similar work. Section 8 reviews the most important works in the field of hierarchical scheduling. Finally, section 9 summarizes the contributions of the article and future lines of work.

2. System model and notation

Our model is concerned with the pre-emptive scheduling of real-time applications on a uniprocessor. Each application consists of a number of *partitions* $P_1, .., P_m$. Each partition comprises a number of tasks. Thus, our hierarchical system has two levels, the partition (or global) level and the task (or local) level, each of them with its own scheduling policy. In this work, we will assume that the local level is scheduled under EDF scheduling policy and the global level is scheduled under any scheduler. The information regarding the global scheduling is provided as temporal windows or slots in which a partition is allowed to execute.

From now on, the sub index used to refer to a partition will be omitted to simplify the notation. Therefore, formally, a partition P can be defined[1] as a tuple $P = \{\tau, R\}$ where:

- $\tau = \{\tau_1, \tau_2, .., \tau_n\}$ is a set of n tasks. A task τ_i is characterized by a tuple $\tau_i = \{\phi_i, C_i, D_i, T_i\}$ where ϕ_i is the offset, C_i is the worst case computation time, D_i is the relative deadline and T_i is the period. When all parameters in the system are integers, we may assume without loss of generality that all preemptions occur at integer time values. We then assume, for the remainder of the article, that all parameters are indeed integers. Moreover, constrained deadlines are assumed so $D_i \leq T_i$.
- An arbitrary CPU supply R is represented by a sequence of p intervals $I_1, I_2, ..., I_p$. Every I_i / $1 \leq i \leq p$ is a closed interval $I_i =: [s_i, e_i]$ repeated every lcm_τ[2], so that $0 \leq s_i < e_i < s_{i+1}$ and $e_p \leq lcm_\tau$.

  Therefore, $\forall t$ exists a unique interval I_i so that $s_i \leq t \leq e_i$. The CPU supply R for a partition determines the p temporal slots in which tasks allocated to the partition are allowed to execute.

The problem to solve is concerns the schedulability of the partition, that is, if task set τ can be scheduled without deadline misses in the slots defined by R.

2.1. Supply bound function

Although we have characterized R as a set of intervals, it can also be represented graphically.

Figure 5 shows two possible CPU supplies for the example of Figure 2 with a periodic supply R=(θ, π), where the global level provides θ units of time each π units. In the figure $\theta = 3$ and $\pi = 10$ so both supplies are non-decreasing functions that grow with a slope of 45 degrees at least 3 units every 10 units. $wcsbf_R(t)$ represents the worst case because provides the 3 units of CPU as late as possible while $sbf_R(t)$ represents other specific allocation of the periodic supply. Therefore, we call the function that represents any specific allocation, the supply bound function of R ($sbf_R(t)$). In this case, note that $sbf_R(t)$ totally coincide with the slots allocation of Figure 2 in the sense that the partition is allowed to execute only when $sbf_R(t)$ function increases.

[1]In the definition of the partition we omit all non-temporal resources

[2]Least Common Multiple of $T_1, .., T_n$

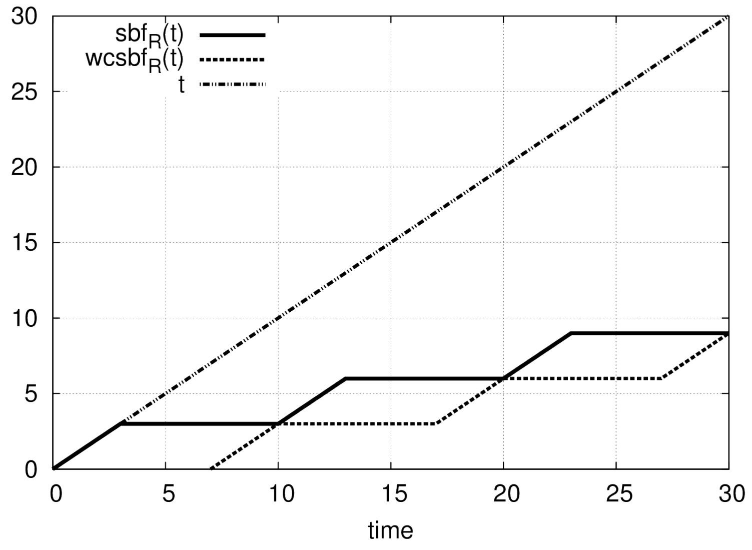


Figure 5: Periodic supply bound functions ($\theta = 3, \pi = 10$)

Given a CPU supply R and an interval of length t, the supply bound function gives the amount of resource that model R is guaranteed to supply in any time interval of length t [5]. We can define the supply bound function of R, accordingly with the above definition.

Definition 1. *The supply bound function ($sbf_R(t)$) of an arbitrary supply R expressed as a set of intervals is:*

$$sbf_R(t) = \begin{cases} \sum_{i=0}^{j}(e_i - s_i) + t - s_j & \textit{if}\, \exists j/t \in [s_j, e_j], \\ \\ \sum_{i=0}^{j}(e_i - s_i) & \textit{if}\, \exists j/e_j < t < s_{j+1}. \end{cases}$$

, where s_i is the starting point of an interval and e_i is its ending point. From s_i to e_i, the tasks of the partition can be executed. In Figure 5, these intervals correspond with the intervals where $sbf_R(t)$ increases. Then, a CPU supply R can be characterized either by a set of intervals I_i or by its $sbf_R(t)$.

Moreover, the following definitions will be used in the next sections.

Definition 2. *[6] The function $G_\tau(t)$ represents the computation time demanded from initial time to time t for a tasks set τ. It can be calculated as:*

$$G_\tau(t) = \sum_{i=1}^{n} C_i \cdot \left\lceil \frac{t}{T_i} \right\rceil$$

It is a positive and non-decreasing function that only increases when a task is released, that is, it grows as many units as time computation are required by the task that has been activated.

If tasks are simultaneously activated at time t = 0 (i.e. $\phi_i = 0$ for all the tasks so the task set is synchronous), then:

Definition 3. *[7][8] The maximum cumulative execution time requested by jobs of τ whose absolute deadlines are less than or equal to t is:*

$$dbf_{\tau}(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor$$

It is a positive and non-decreasing function that only increases in the so-called *scheduling points* that is, when a deadline arrives.

To generalize, when the task set is asynchronous (i.e. $\exists \phi_i \neq 0$), the processor demand function in interval $[t_1, t_2)$ is defined as:

Definition 4. *[9][8]*

$$dbf(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1, t_2) C_i$$

where:

$$\eta_i(t_1, t_2) = max\{0, (\left\lfloor \frac{t_2 - \phi_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \phi_i}{T_i} \right\rceil + 1)\}$$

From now on, let us assume the task set is synchronous. Therefore, definition 3 will be used to deduct the minimum supply bound function, in spite of the possibility of using definition 4 to obtain any other demand function.

3. Schedulable supply bound functions

In this section, we will define specific $sbf_R(t)$ that ensure the schedulability of task sets, τ. Specifically, two functions are obtained: $gsbf_\tau(t)$ and $msbf_\tau(t)$.

3.1. Schedulable $sbf_\tau(t)$ based on G(t)

This section presents the demanded computation supply function ($gsbf_\tau(t)$). This function gives a schedulable supply for τ. We will base our method on the $G_\tau(t)$ function (Definition 2).

Definition 5. *A characteristic point, t_j, of $G_\tau(t)$ is the one complying with:*

$$G_\tau(t_j - \epsilon) < G_\tau(t_j + \epsilon) \quad 0 \leq t_j \leq lcm_\tau \quad \forall \epsilon \to 0$$

, so that t_j coincides with the activation of $\tau_i \in \tau$.

Property 1. *[7] Let τ be a schedulable task set. Let t_x be an instant t_x such that:*

$$G_\tau(t_x) \leq t_x$$

Then, the processor must have been idle for at least $t_x - G_\tau(t_x)$ time units from initial time.

From Definition 2 and Property 1, the demanded computation supply function ($gsbf_\tau(t)$) is presented.

Definition 6. *The $gsbf_\tau(t)$ is defined as:*

$$gsbf_\tau(t) = \begin{cases} t - \sum_{i=0}^{j-1}(s_{i+1} - e_i) & \text{if } \exists j / t \in [s_j, e_j], \\ \\ G_\tau(t) & \text{if } \exists j / e_j < t < s_{j+1}. \end{cases}$$

, where:

$$\begin{aligned} s_j &= t_j + \theta_j \\ e_j &= t_j - G_\tau(t_j - \epsilon) + G_\tau(t_j + \epsilon) + \theta_j \\ \theta_j &= max\{0, (e_{j-1} - t_j)\} \end{aligned}$$

and each t_j is an characteristic point of $G_\tau(t_j)$.

Figure 6 shows how the function is obtained graphically in the first intervals. In this Figure, the characteristic points are depicted ($t_1, t_2, ...$) and, applying Definition 6, the start and end points of the intervals of $gsbf_\tau(t)$ are calculated. They correspond to the intervals where $gsbf_\tau(t)$ increases.

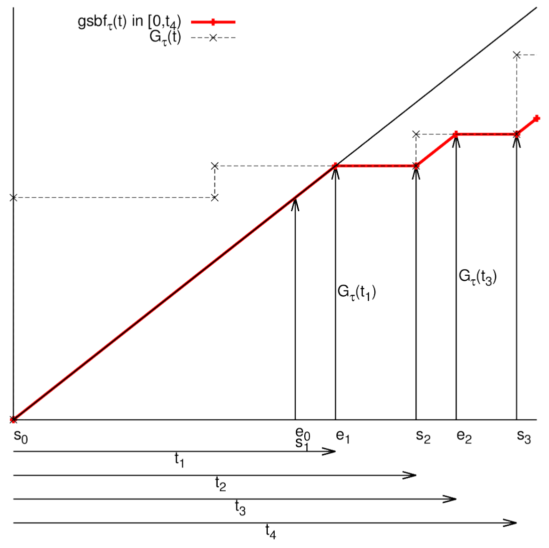


Figure 6: Calculation of $gsbf_\tau(t)$ in $[0, t_4)$

To obtain a more compact definition, let us replace the values of s_j and e_j:

$$\begin{aligned}\sum_{i=0}^{j-1}(s_{i+1}-e_i) = (s_1-e_0)+(s_2-e_1)+...+(s_j-e_{j-1})\\ &= t_1+\theta_1-(t_0-G_\tau(t_0-\epsilon)+G_\tau(t_0+\epsilon)+\theta_0+\\ &+t_2+\theta_2-(t_1-G_\tau(t_1-\epsilon)+G_\tau(t_1+\epsilon)+\theta_1+\\ &+...+\\ &+t_{j-1}+\theta_{j-1}-(t_{j-1}-G_\tau(t_{j-2}-\epsilon)+\\ &+G_\tau(t_{j-2}+\epsilon)+\theta_{j-2})+\\ &+t_j+\theta_j-(t_j-G_\tau(t_{j-1}-\epsilon)+G_\tau(t_{j-1}+\epsilon)+\theta_{j-1})+\\ &=-t_0+G_\tau(t_0-\epsilon)-\theta_0+t_j+\theta_j-G_\tau(t_{j-1}+\epsilon)\end{aligned}$$

As $t_0 = G_\tau(t_0-\epsilon) = \theta_0 = 0$ and $G_\tau(t_{j-1}+\epsilon) = G_\tau(t_j-\epsilon)$, then, one more compact definition of $gsbf_\tau(t)$ is:

$$gsbf_\tau(t) = \begin{cases} t-t_j-\theta_j+G_\tau(t_j-\epsilon) & \text{if } \exists j/t \in [s_j, e_j], \\ \\ G_\tau(t) & \text{if } \exists j/e_j < t < s_{j+1}. \end{cases}$$

Once $gsbf_\tau(t)$ has been defined, the schedulability of τ under this CPU supply can be demonstrated. For this reason, the concept of initial critical interval (ICI) must be introduced. ICI [10] is defined as the temporal interval between initial time and the first instant R, when all requests have already been served and no additional requests have arrived yet, assuming that the processor is not idle while tasks are pending. Therefore, this range is [0, R).

Lemma 1. *[10] [11] Let τ be a synchronous periodic task set and [0,R) its initial critical interval (ICI). τ is schedulable if and only if it can be scheduled in ICI.*

Next theorems are used to construct the $gsbf_\tau(t)$ interval by interval and, in each interval, the condition $gsbf_\tau(t) \leq t$ will be checked.

Theorem 1. *Let $t_1 \in \mathbb{N}$ so that:*

$$\min_t t_1 : G_\tau(t_1) \leq t_1 \quad \forall t \in (0, lcm_\tau]$$

Then,

$$gsbf(t) = t \quad \textit{if } t \in [0, t_1)$$

Proof. Because of the definition of ICI, in the range [0,R) there is no CPU idle time. For this reason, from 0 to t_1, the processor must be always busy. So, $t_1 = R$.

Theorem 2. *Let $t_2 \in \mathbb{N}$ so that:*

$$\min_t t_2 : G_\tau(t_2 - \epsilon) \leq G_\tau(t_2 + \epsilon) \quad t_1 < t_2 \leq lcm_\tau \quad \forall \epsilon \to 0$$

, that is, t_2 is a characteristic point of $G_\tau(t)$.

Then,

$$gsbf_\tau(t) = \begin{cases} t & \textit{if } t \in [0, t_1), \\ G_\tau(t) & \textit{if } t \in [t_1, t_2). \end{cases}$$

Proof. The proof is based on the addition of a new task τ_{n+1}, which will be executed only when CPU is idle, that is, when τ is not executing. This time corresponds to the interval $[t_1, t_2)$. Let

$$\tau' = \tau \bigcup \tau_{n+1}$$

where

$$\begin{aligned} \phi_{n+1} &= t_1 \\ C_{n+1} &= t_2 - t_1 \\ D_{n+1} &= t_2 \\ T_{n+1} &= max\{\phi_1, ..., \phi_N, \phi_{n+1}\} + 2 \cdot lcm_\tau [9] \\ &= \phi_{n+1} + 2 \cdot lcm_\tau \end{aligned}$$

Once this task has been added, let's check that the first interval, R, when all requests have already been served and no additional requests have arrived yet, is $[0, t_2)$.

$$\begin{aligned} G_{\tau'}(t) &= G_\tau(t) + G_{\tau^{n+1}}(t) \\ &= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) \left\lceil \frac{t}{\phi_{n+1} + 2 \cdot lcm_\tau} \right\rceil \\ &= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) \end{aligned}$$

When $t = t_2 - \epsilon$, $G_\tau(t) = t_1$ the result of the previous equation is:

$$\begin{aligned}
G_{\tau'}(t_2 - \epsilon) &= G_\tau(t_2 - \epsilon) + G_{\tau^{n+1}}(t_2 - \epsilon) \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t_2 - \epsilon}{T_i} \right\rceil + (t_2 - t_1) \\
&= t_1 + (t_2 - t_1) \\
&= t_2
\end{aligned}$$

It is clear that, adding this new task set, $G_{\tau'}(t) = t$ and the new ICI is $[0, t_2)$. So, as a result of Lemma 1, to prove schedulability of τ', the condition $dbf_{\tau'}(t) \leq t$ must be held in all the scheduling points in $[0, t_2)$.

Let us assume that t_x is a scheduling point in $[0, t_2)$. As $t_x < D_{n+1}$ clearly $dbf_{\tau'}(t_x) = dbf_\tau(t_x) \leq t_x$. Therefore, the schedulability of τ has been demostrated because of its schedulability in ICI.

As a result of the previous theorem, we derive the next interval.

Lemma 2. *Let* $t_3, t_4 \in \mathbb{N}$ *so that:*

$$\begin{aligned}
t_3 &: G_\tau(t_2 + \epsilon) - G_\tau(t_2 - \epsilon) + t_2 \\
t_4 &: G_\tau(t_4 - \epsilon) \leq G_\tau(t_4 + \epsilon)
\end{aligned}$$

, where $t_2 \leq t_3 \leq t_4 \leq lcm_\tau, \quad \forall \epsilon \to 0$

Then,

$$gsbf_\tau(t) = \begin{cases} t & \textit{if } t \in [0, t_1), \\ G_\tau(t) & \textit{if } t \in [t_1, t_2), \\ t - (t_2 - t_1) & \textit{if } t \in [t_2, t_3), \\ G_\tau(t) & \textit{if } t \in [t_3, t_4). \end{cases}$$

PROOF. Following the same reasoning as Theorem 2, we add a new task whose computation time coincides:

$$\tau'' = \tau \bigcup \tau_{n+1} \bigcup \tau_{n+2}$$

where

$$\begin{aligned}
\phi_{n+1} &= t_1 \\
C_{n+1} &= t_2 - t_1 \\
D_{n+1} &= t_2 \\
T_{n+1} &= max\{\phi_1, ..., \phi_N, \phi_{n+1}, \phi_{n+2}\} + 2 \cdot lcm_\tau \\
&= \phi_{n+2} + 2 \cdot lcm_\tau
\end{aligned}$$

$$\begin{aligned}
\phi_{n+2} &= t_3 \\
C_{n+2} &= t_4 - t_3 \\
D_{n+2} &= t_4 \\
T_{n+2} &= max\{\phi_1, ..., \phi_N, \phi_{n+1}, \phi_{n+2}\} + 2 \cdot lcm_\tau \\
&= \phi_{n+2} + 2 \cdot lcm_\tau
\end{aligned}$$

Once this task has been added, let us calculate the ICI in this new scenario:

$$\begin{aligned}
G_{\tau''}(t) &= G_{\tau}(t) + G_{\tau^{n+1}}(t) + G_{\tau^{n+2}}(t) \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) \left\lceil \frac{t}{\phi_{n+2} + 2 \cdot lcm_{\tau}} \right\rceil + \\
&+ (t_4 - t_3) \left\lceil \frac{t}{\phi_{n+2} + 2 \cdot lcm_{\tau}} \right\rceil \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t}{T_i} \right\rceil + (t_2 - t_1) + (t_4 - t_3)
\end{aligned}$$

When $t = t_4 - \epsilon$, $G_{\tau}(t) = t_1 + (t_3 - t_2)$. Therefore:

$$\begin{aligned}
G_{\tau''}(t_4 - \epsilon) &= G_{\tau}(t_4 - \epsilon) + G_{\tau^{n+1}}(t_4 - \epsilon) + G_{\tau^{n+2}}(t_4 - \epsilon) \\
&= \sum_{i=1}^{n} C_i \left\lceil \frac{t_4 - \epsilon}{T_i} \right\rceil + (t_2 - t_1) + (t_4 - t_3) \\
&= t_1 + (t_3 - t_2) + (t_2 - t_1) + (t_4 - t_3) \\
&= t_4
\end{aligned}$$

It can be concluded that, adding a new task set,τ_{n+2}, $G_{\tau''}(t) = t$ in $t = t_4$ so the new ICI is $[0, t_4)$. So, as a result of Lemma 1, to prove schedulability of τ'', the condition $dbf_{\tau''}(t) \leq t$ must be held in all the scheduling points in $[0, t_4)$.

Let us assume that t_x is a scheduling point in $[0, t_4)$. As $t_x < D_{n+2}$, in Theorem 2 it has been demonstrated that $dbf_{\tau''}(t_x) = dbf_{\tau'}(t_x) \leq t_x$. So, τ'' is schedulable in the hyperperiod because of its schedulability in ICI.

From these first intervals, the complete definition of the demanded computation supply function can be built recursively. Values of s_j and e_j are deduced according to the different shapes of the function, depending on how $G_{\tau}(t)$ is built.

The algorithm that implements the slot construction is presented in Listing 1.

Listing 1: $gsbf_{\tau}(t)$ algorithm

```
function gsbf($\tau$) is
 $j$,$e_j$,$s_j$,$t_0$,$t_1$=0;
 $\epsilon \to 0$;
 while($t_1$<$lcm_{\tau}$) loop
    if $G(t_0 - \epsilon) < G(t_0 + \epsilon)$ then
          $\theta_0 = max\{0, e_j - t_0\}$;
          $s_j = t_0 + \theta_0$;
          $e_j = t_0 - G(t_0 - \epsilon) + G(t_0 + \epsilon) + \theta_0$;
    end if ;
    $t_0$++; $j$++;
 end while;
end gsbf;
```

The $gsbf_\tau(t)$ represents a set of temporal windows that can successfully schedule τ. Many sets of slots fulfil this purpose, however, because CPU time is supplied only when a task is activated, that is, as soon as possible. And for this reason, the resulting schedule exactly coincides with the schedule resulting from assigning all CPU time to the partition.

3.1.1. Example of $gsbf_\tau(t)$ use

Let's consider a partition with three tasks ($\tau = \{\tau_1, \tau_2, \tau_3\}$). Task parameters are listed in Table 1. The definition of $gsbf_\tau(t)$ in definition 6 is used to calculate the

Table 1: Task parameters

	C_i	D_i	T_i
τ_0	1	4	5
τ_1	6	10	15
τ_2	5	21	30

function. The first step consists in calculating $G_\tau(t)$ and all its characteristic points in $[0, lcm_\tau]$. Table 2 shows all these values. Now, applying Definition 6, s_i and e_i

Table 2: Characteristic points of $gsbf_\tau(t)$

t	0	5	10	15	20	25
$G_\tau(t)$	12	13	14	21	22	23

are calculated within $[0, lcm_\tau]$ and represented in Table 3. The last step is to calcu-

Table 3: Definition of $[s_i, e_i]$

i	0	1	2	3	4	5
s_i	0	12	13	15	22	25
e_i	12	13	14	22	23	26

late $gsbf_\tau(t)$ inside each interval. Once it has been calculated, the representation of the function is shown in Figure 7. As seen, CPU will be busy in the intervals where $gsbf_\tau(t)$ grows. These intervals are defined in Table 4. The execution chronogram of the task set considering that the CPU supply R coincides with the demanded computation supply function is depicted in Figure 8. As seen, the task set is schedulable in this slot assignment.

Thus, the schedulability of the $gsbf_\tau(t)$ function for a task set and the methodology for calculating it have been demonstrated.

3.2. Schedulable $sbf_\tau(t)$ based on $dbf_\tau(t)$

In Section 3.1, a valid supply that gives CPU when tasks are activated has been presented. Now, another valid supply is being presented but, in this new situation, CPU is supplied just before the deadlines arrive. If $gsbf_\tau(t)$ consists on supplying

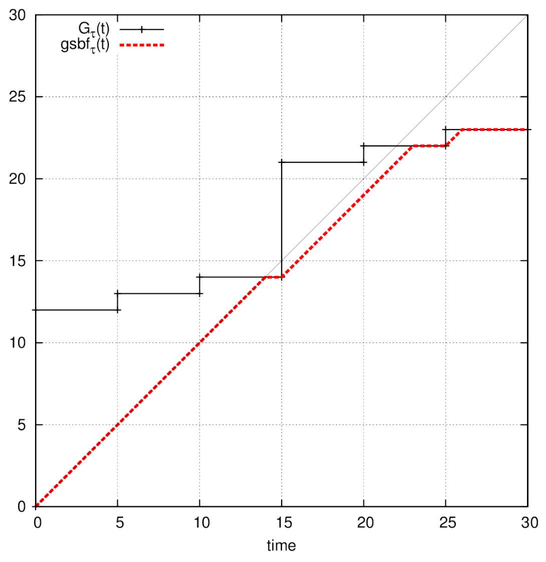


Figure 7: Representation of $gsbf_\tau(t)$

Table 4: Definition of $[s_i, e_i]$

	s_i	e_i
I_1	0	14
I_2	15	23
I_3	25	26

CPU as soon as possible (for task activation), $msbf_\tau(t)$ will consist in supplying CPU as late as possible.

To obtain $msbf_\tau(t)$, we base our method on the demand bound function for a task set. We build $msbf_\tau(t)$ by intervals and we prove that in each interval there are no deadline misses. Thus, $msbf_\tau(t)$ will be generalized.

Theorem 3. *Let $t_1 \in \mathbb{N}$ so that:*

$$t_1 - dbf_\tau(t_1) = \min_t(t - dbf_\tau(t)) \quad \forall t \in (0, lcm_\tau]$$

And,

$$msbf_\tau(t) = \begin{cases} t - t_1 + dbf_\tau(t_1) & \textit{if } t \in [t_1 - dbf_\tau(t_1), t_1], \\ 0 & \textit{if } t \in [0, t_1 - dbf_\tau(t_1)). \end{cases}$$

If $sbf_R(t) = msbf_\tau(t)$ then τ is schedulable.

PROOF. The proof is based on adding a new task τ_{n+1}. This task can only be executed when τ is not allowed to execute, that is, in $[0, t_1 - dbf_\tau(t_1))$ and we demonstrate that

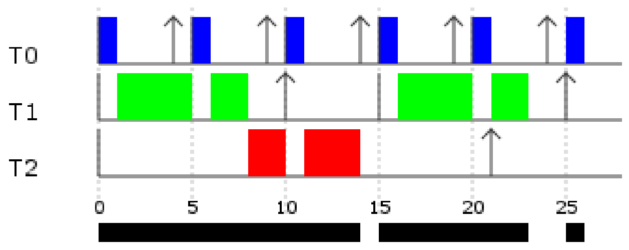


Figure 8: Execution chronogram of τ with $gsbf_\tau(t)$ in Table 4

the new set is schedulable. Then, computation time of τ_{n+1} is increased in order to check that the new set is not schedulable.

Let

$$\tau' = \tau \bigcup \tau_{n+1}$$

where

$$\begin{aligned} C_{n+1} &= t_1 - dbf_\tau(t_1) \\ D_{n+1} &= t_1 - dbf_\tau(t_1) \\ T_{n+1} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+1} &= 0 \end{aligned}$$

To prove schedulability of τ', the condition $dbf_{\tau'}(t) \leq t$ must be met in all the scheduling points in $[0, t_1]$. Let us assume that a is a scheduling point in $[0, t_1]$. If $a < D_{n+1}$, obviously $dbf_{\tau'}(a) = dbf_\tau(a) \leq a$. If $a \geq D_{n+1}$ the demand bound function of the new task set τ' is:

$$\begin{aligned} dbf_{\tau'}(a) &= dbf_\tau(a) + C_{n+1} \\ &= dbf_\tau(a) + t_1 - dbf_\tau(t_1) \end{aligned}$$

As $t_1 - dbf_\tau(t_1) = \underset{t}{min}(t - dbf_\tau(t))$ then

$$t_1 - dbf_\tau(t_1) \leq (a - dbf_\tau(a))$$

So,

$$\begin{aligned} dbf_{\tau'}(a) &\leq dbf_\tau(a) + a - dbf_\tau(a) \\ &\leq a \end{aligned}$$

Now, let us assume

$$\tau'' = \tau \bigcup \tau_{n+1}$$

and

$$\begin{aligned} C_{n+1} &= t_1 - dbf_\tau(t_1) + \epsilon \\ D_{n+1} &= t_1 - dbf_\tau(t_1) + \epsilon \\ T_{n+1} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+1} &= 0 \end{aligned}$$

being ϵ a small positive number such that $0 < \epsilon \leq 1$.
Following the same reasoning:

$$\begin{aligned} dbf_{\tau'}(t_1) &= dbf_\tau(t_1) + t_1 - dbf_\tau(t_1) + \epsilon \\ &= t_1 + \epsilon \end{aligned}$$

so τ'' is not schedulable.

As a result of the previous theorem, $msbf_\tau(t)$ until t_1, expressed as a set of intervals, is $msbf_\tau(t) = I_0 = [t_1 - dbf_\tau(t_1), t_1]$. Using a similar approach we derive the next interval.

Lemma 3. *Let $t_2 \in \mathbb{N}$, $t_1 < t_2$ so that:*

$$t_2 - dbf_\tau(t_2) = \min_t(t - dbf_\tau(t)) \quad \forall t \in (t_1, lcm_\tau]$$

And

$$msbf_\tau(t) = \begin{cases} t - t_1 + dbf_\tau(t_1) & \textit{if}\, t \in [t_1 - dbf_\tau(t_1), t_1], \\ dbf_\tau(t_1) & \textit{if}\, t \in (t_1, \\ & t_2 - dbf_\tau(t_2) + dbf_\tau(t_1)) \\ t - t_2 + dbf_\tau(t_2) & \textit{if}\, t \in [t_2 - dbf_\tau(t_2) \\ & + dbf_\tau(t_1), t_2], \\ 0 & \textit{if}\, t \in [0, t_1). \end{cases}$$

If $sbf_R(t) = msbf_\tau(t)$ then τ is schedulable.

PROOF. Schedulability in $[0, t_1]$ is assured due to Theorem 3. Following the same reasoning as Theorem 3, we add a task whose computation time coincides with the idle time between I_0 and SI_1 and its deadline is equal to the start time of I_1 (s_1).

Let

$$\tau' = \tau \bigcup \tau_{n+1} \bigcup \tau_{n+2}$$

where

$$\begin{aligned} C_{n+1} &= t_1 - dbf_\tau(t_1) \\ D_{n+1} &= t_1 - dbf_\tau(t_1) \\ T_{n+1} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+1} &= 0 \end{aligned}$$

$$\begin{aligned} C_{n+2} &= t_2 - dbf_\tau(t_2) + dbf_\tau(t_1) - t_1 \\ D_{n+2} &= t_2 - dbf_\tau(t_2) + dbf_\tau(t_1) \\ T_{n+2} &= max\{\phi_1, ..., \phi_N\} + 2 \cdot lcm_\tau \\ \phi_{n+2} &= t_1 \end{aligned}$$

Let us assume that a is a scheduling point in $(t_1, t_2]$. If $a < D_{n+2}$, then $dbf_\tau(a) = dbf_{\tau'}(a)$, so the new task set is schedulable. If $a \geq D_{n+2}$, following the same reasoning as in Theorem 3, the computation time of τ_{n+1} and τ_{n+2} is added to $sbf_{\tau'}(t)$:

$$\begin{aligned} dbf_{\tau'}(a) &= dbf_\tau(a) + t_1 - dbf_\tau(t_1) + \\ &\quad + t_2 - dbf_\tau(t_2) + dbf_\tau(t_1) - t_1 \\ &= dbf_\tau(a) + t_2 - dbf_\tau(t_2) \end{aligned}$$

Given

$$t_2 - dbf_\tau(t_2) \leq (a - dbf_\tau(a))$$

that

$$\begin{aligned} dbf_{\tau'}(a) &\leq dbf_\tau(a) + a - dbf_\tau(a) \\ &\leq a \end{aligned}$$

Theorem 3 and Lemma 3 provide a method for obtaining the first two intervals of $msbf_\tau(t)$ function and it has been proved that this function is schedulable. Figure 9 shows graphically how the function is obtained.

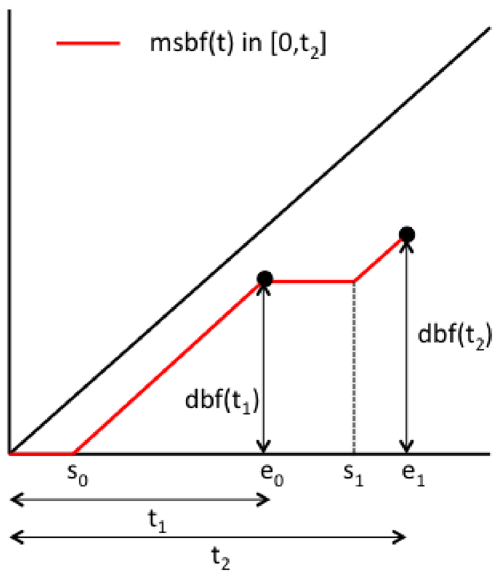


Figure 9: Calculation of $msbf_\tau(t)$ in $[0, t_2]$

It is straightforward to recursively construct all the minimum supply slots needed by τ to maintain feasibility: finding t_j points in where it holds that $t_j - dbf_\tau(t_j)$ is the minimum value in $(t_{j-1}, lcm_\tau]$. We call these points the *minimum scheduling points*

t_j. Therefore, the $msbf_\tau(t)$ is defined as in Definition 1 but now we can give specific values to s_j and e_j:

$$msbf_\tau(t) = \begin{cases} \sum_{i=0}^{j}(e_i - s_i) + t - e_j & \text{if } \exists j/t \in [s_j, e_j], \\ \\ \sum_{i=0}^{j}(e_i - s_i) & \text{if } \exists j/e_j < t < s_{j+1}. \end{cases}$$

where $s_j = t_j - dbf_\tau(t_j) + dbf(t_{j-1})$ and $e_j = t_j$.
Replacing the values of s_j and e_j in the previous definition:

$$\sum_{i=0}^{j}(e_i - s_i) = t_1 - t_1 + dbf_\tau(t_1) - dbf(t_0) +$$

$$+ t_2 - t_2 + dbf_\tau(t_2) - dbf(t_1) + ...$$

Assuming that $t_0 = 0$ and $dbf_\tau(0) = 0$:

$$\sum_{i=0}^{j}(e_i - s_i) = dbf_\tau(t_j)$$

Therefore, we can provide a more compact definition for $msbf_\tau(t)$.

Definition 7. *The definition of the minimum supply bound function* $msbf_\tau(t)$ *is:*

$$msbf_\tau(t) = \begin{cases} t - t_j + dbf_\tau(t_j) & \textit{if } \exists j/t \in [s_j, e_j], \\ \\ dbf_\tau(t_j) & \textit{if } \exists j/e_j < t < s_{j+1}. \end{cases}$$

The algorithm that implements the slot construction is presented in Listing 2.

Listing 2: $msbf_\tau(t)$ algorithm

```
function msbf($\tau$) is
 i, $e_i$,$s_i$,$t_1$,$t_2$=0;
 while($t_2$<$lcm_\tau$) loop
     $t_2$= $\min_{e_i<t\le lcm_\tau}(dbf_\tau(t))$;
     $s_i = t_2 - dbf_\tau(t_2) + dbf_\tau(t_1)$;
     $e_i = t_2$;
     $t_1 = t_2$;
     i++;
 end while;
end msbf;
```

The previous function is obtained from $dbf_\tau(t)$ in Definition 3, particularized for synchronized tasks. If we assume the possibility of asynchronism between tasks (i.e.,

$\phi_i \neq 0$), this algorithm is also valid due to the inclusion of the offset in Definition 4. If a task set is synchronous, Definition 3 will be applied to obtain $dbf_\tau(t)$ and, consequently, $msbf_\tau(t)$. However, if any task does not start at the same time as others, then another $dbf_\tau(t)$ will be obtained because of this offset and, consequently, other $msbf_\tau(t)$, which will also meet the criteria of schedulability of all tasks.

The previous algorithm works over the entire hyperperiod (lcm_τ), which depending on the values of task periods can be a large value. To overcome this disadvantage, we can use the results presented by Brocal and Balbastre In [12], an algorithm is presented that can be used to compute the minimum hyperperiod for a set of periodic activities when period is specified as a range. If, in spite of considering periods as specific values, we treat them as ranges of valid values, [12] will select the value inside each interval which causes the minimum hyperperiod. This method drastically reduces the hyperperiod.

As noted in the conclusions section, we are working on an upper bound of $msbf(\tau)$ to reduce the complexity of the algorithm.

Once $msbf_\tau(t)$ is obtained, the following theorem provides the schedulability condition of $\{\tau, R\}$.

Theorem 4. *A task set τ is schedulable under a CPU supply R if and only if:*

$$\forall t \quad sbf_R(t) = msbf_\tau(t)$$

PROOF. We prove that for any time point a :

$$msbf_\tau(a) \geq dbf_\tau(a) \quad \forall a \in [0, lcm_\tau]$$

We assume two cases:

- Case 1: $a \notin [s_j, e_j]$.
- Case 2: $a \in [s_j, e_j]$.

Case 1: If $a \notin [s_j, e_j]$, then $\exists j$ so $e_j < a < s_{j+1}$. Applying the second case in the $msbf_\tau(t)$ definition:

$$msbf_\tau(a) = dbf_\tau(a)$$

Case 2: If $a \in [s_j, e_j]$, applying the first case of the msbf definition:

$$msbf_\tau(a) = a - t_j + dbf_\tau(t_j)$$

As $dbf_\tau(t)$ is a positive and monotonic increasing function it holds that [10]:
If $a \leq t_j$ then $dbf_\tau(a) \leq dbf_\tau(t_j)$
And, as τ is schedulable then $dbf_\tau(t_j) - t_j \leq 0$.
Therefore:

$$\begin{aligned} msbf_\tau(a) &\geq dbf_\tau(a) - dbf_\tau(t_j) - t_j \\ &\geq dbf_\tau(a) \end{aligned}$$

In any case: $msbf_\tau(a) \geq dbf_\tau(a)$.

3.2.1. Example of $msbf_\tau(t)$ use

Let us consider a partition with three tasks ($\tau = \{\tau_1, \tau_2, \tau_3\}$). Task parameters are listed in Table 1. Figure 10 shows the execution chronogram for the task set scheduled under EDF policy if the partition is the only one in the system, that is, the CPU supply is a unique slot $I_0 = [0, 30]$. As the figure shows, the task set is schedulable since there are no missed deadlines throughout the hyperperiod (lcm_τ).

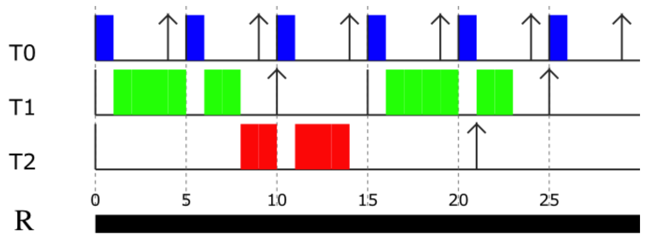


Figure 10: Execution chronogram of τ with $R = I_0 = [0, 30]$

Let us consider now that there are more partitions in the system so the SI assigns to the considered partition a CPU supply R which matches the $msbf_\tau(t)$.

To find out whether the CPU supply R is able to successfully schedule the task set, we obtain $msbf_\tau(t)$. The methodology, as explained in the previous sections consists of calculating the minimum $t - dbf_\tau(t)$ in $[0, lcm_\tau]$. Table 5 shows $t - dbf_\tau(t)$ for all the scheduling points in [0,30].

Table 5: Scheduling points

t	4	9	10	14	19	21	24	25	29
$dbf_\tau(t)$	1	2	8	9	10	15	16	22	23
$t - dbf_\tau(t)$	3	7	2	5	9	6	8	3	6

The scheduling point with the minimum slack ($t - dbf_\tau(t)$) is $t_1 = 10$. Therefore, according to Theorem 3, the first slot of $msbf_\tau(t)$ is $I_0 = [t_1 - dbf_\tau(t_1), t_1] = [2, 10]$.

In the second iteration, we must search for the next scheduling point with the minimum slack in (10,30]. This point is $t_2 = 25$. Therefore, the next slot of $msbf_\tau(t)$ is $I_1 = [t_2 - dbf_\tau(t_2) + dbf_\tau(t_1), t_2] = [11, 25]$.

In the third and last iteration, it is clear that $t_3 = 29$ as it is the only remaining scheduling point in $(25, 30]$. Therefore $I_2 = [t_3 - dbf_\tau(t_3) + dbf_\tau(t_2), t_3] = [28, 29]$. Table 6 summarizes the slots of $msbf_\tau(t)$ and the representation of the function is depicted in Figure 11.

Table 6: Minimum supply bound

	s_i	e_i
I_1	2	10
I_2	11	25
I_3	28	29

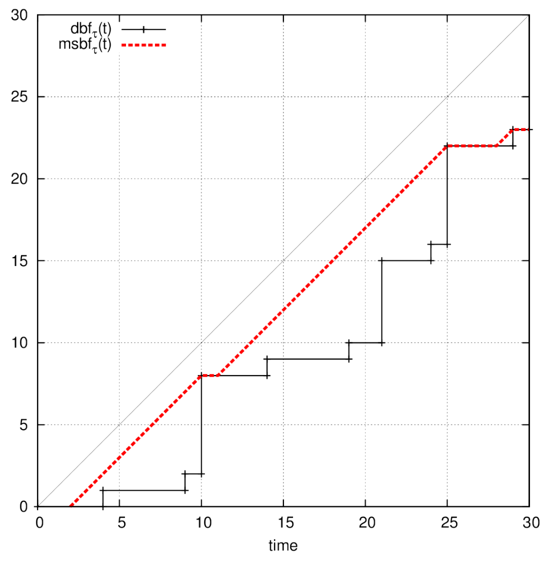


Figure 11: Representation of $msbf_{\tau}(t)$

The execution chronogram of the task set considering that the CPU supply R coincides with the minimum supply calculated is depicted in Figure 12. We see in the figure that the task set is schedulable and that this slots assignment are the more restrictive ones, since if any slot is reduced, task τ_2 will miss its deadline.

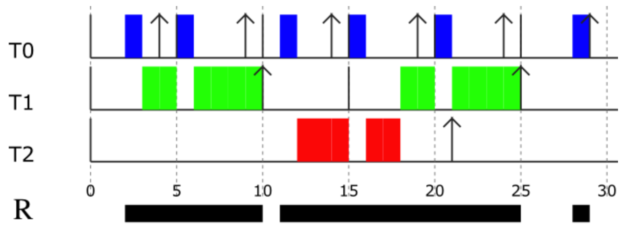


Figure 12: Execution chronogram of τ with $R = msbf_{\tau}(t)$

As $sbf_R(t) = msbf_{\tau}(t)$, the task set is schedulable. It is depicted in Figure 12. Finally, it has become clear the schedulability analysis exposed in section 5.

4. Schedulability areas

In this section, we obtain information about the schedulability of $sbf_R(t)$ depending on its relationship with $gsbf_\tau(t)$ and $msbf_\tau(t)$.

Both $gsbf_\tau(t)$ and $msbf_\tau(t)$ are minimum supply functions in the sense that both supply the minimum instants of CPU to assure schedulability of τ.

In Figure 13, three zones according to the relation with $msbf_\tau(t)$ and $gsbf_\tau(t)$ are depicted:

- Zone 1: This area depicts CPU supply R functions that are greater than $gsbf_\tau(t)$ and, consequently, greater than $msbf_\tau(t)$.
- Zone 2: This area depicts CPU supply R functions that are between $msbf_\tau(t)$ and $gsbf_\tau(t)$.
- Zone 3: This area depicts CPU supply R functions that are lesser than $msbf_\tau(t)$.

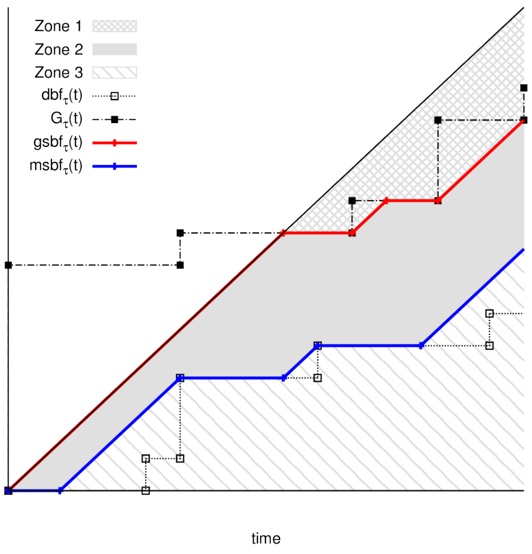


Figure 13: Zones according to the position of $msbf_\tau(t)$ and $gsbf_\tau(t)$

For the periodic resource model with D=T (periodic $sbf_R(t)$) any $sbf_R(t)$ located in Zone 1 and 2 is schedulable whereas any $sbf_R(t)$ located in Zone 3 is not ([13]). However, as the following counterexamples show, this is no longer true if the slot assignation is arbitrary.

4.1. ZONE 1: CPU supply R greater than $gsbf_\tau(t)$ and, consequently, $msbf_\tau(t)$

Some might think that any sequence of slots whose characteristic function $sbf_R(t)$ is greater than $gsbf_\tau(t)$ and, consequently, $msbf_\tau(t)$, would assure the schedulability of task set τ. The next example is used to refute this theory.

Let us consider the partition whose tasks are defined in Table 1. Let us consider now that there are more partitions in the system so the SI assigns to the considered partition a CPU supply R shown in Table 7.

Table 7: CPU supply R

	s_i	e_i
I_1	0	5
I_2	7	25
I_3	29	30

The $msbf_\tau(t)$ of the task set is calculated following the description in Section 3.2 and, representing in the same graph $msbf_\tau(t)$ and $sbf_R(t)$, it is observed that $sbf_R(t)$ is always above or equal to $msbf_\tau(t)$ (see Figure 14).

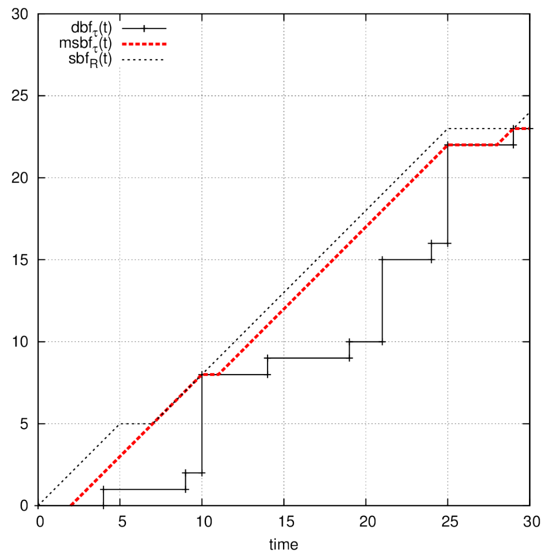


Figure 14: Graphical representation of $dbf_\tau(t)$, $msbf_\tau(t)$ and $sbf_R(t)$

However, as can be seen in Figure 15, this task set is not schedulable in partition R defined by $sbf_R(t)$. τ_0 misses its deadline in its fifth activation. τ_0 is activated in $time = 25u.t.$ and the next deadline is in $time = 29u.t.$. Inside the interval [25,29], there is no CPU supply R. So, the task set is not schedulable.

Although $sbf_R(t)$ is above $msbf_\tau(t)$, the partition must be defined in such a way as to guarantee the that every task meets the deadlines. If, from the request of a task until the arrival of the corresponding deadline, the task does not have enough time to being executed entirely, the system will not be schedulable, as occurs in the previous situation.

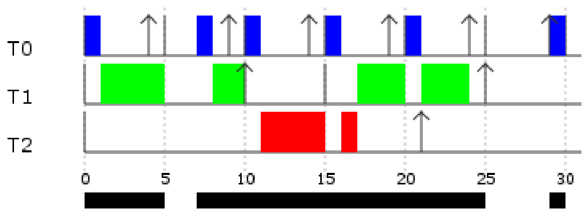


Figure 15: Execution chronogram of τ with R in Table 7

The same happens with $gsbf_\tau(t)$. Therefore, the idea of ensuring the schedulability of a task set whenever $sbf_R(t)$ is greater than $gsbf_\tau(t)$ is rejected.

Let us consider the task set in Table 1 and the CPU supply in Table 8. As Figure 16 shows, $sbf_{R'}(t)$ is always greater than or equal to $gsbf_\tau(t)$ but, as Figure 17 shows, the task set is not schedulable.

Table 8: CPU supply R'

	s_i	e_i
I_1	0	25
I_2	29	30

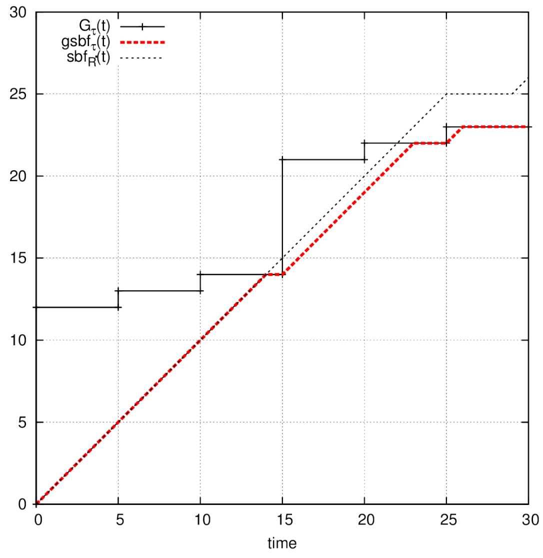


Figure 16: Graphical representation of $G_\tau(t)$, $gsbf_\tau(t)$ and $sbf_{R'}(t)$

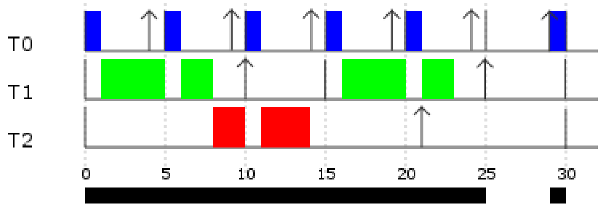


Figure 17: Execution chronogram of τ with R' in Table 8

4.2. *ZONE 2: CPU supply R between $msbf_\tau(t)$ and $gsbf_\tau(t)$*

Now, let us suppose another scenario: $msbf_\tau(t) \leq sbf_R(t) \leq gsbf_\tau(t)$ used to check that one $sbf_R(t)$, which is defined between $gsbf_\tau(t)$ and $msbf_\tau(t)$, does not necessarily assure the schedulability of the task set. It depends on how the function is defined and if the deadlines are met or not.

Let us define a partition defined by the tasks τ' in Table 9 and the CPU supply R" in Table 10.

Table 9: Task parameters τ'

	C_i	D_i	T_i
τ'_0	2	8	10
τ'_1	5	10	25
τ'_2	7	40	50

Table 10: CPU supply R"

	s_i	e_i
I_1	2	16
I_2	21	25
I_3	32	39
I_4	43	44
I_5	45	46

Calculating all the functions as described in previous sections, Figure 18 shows $msbf_{\tau'}(t) \leq sbf_{R''}(t) \leq gsbf_{\tau'}(t)\ \forall t$. However, Figure 19 shows that τ' is not schedulable, because τ'_1 misses its deadline in its second activation.

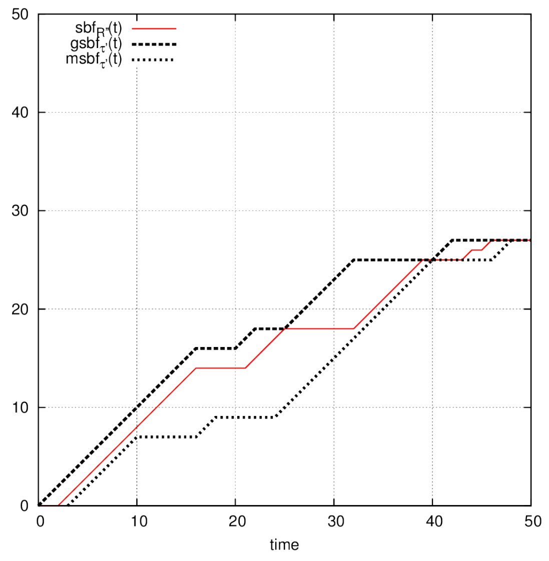


Figure 18: Graphical representation of $msbf_{\tau'}(t)$, $gsbf_{\tau'}(t)$ and $sbf_{R''}(t)$

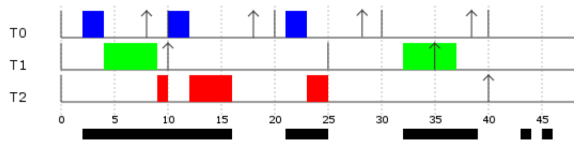


Figure 19: Execution chronogram of τ' with R'' in Table 10

4.3. ZONE 3: CPU supply R less than $msbf_\tau(t)$

According to the definition of $msbf_\tau(t)$, it is the most restrictive function for scheduling a task set. If any supply CPU R is lesser than $msbf_\tau(t)$, the task set will not be schedulable.

Let us define now a CPU supply, R"', shown in Table 11 and task set τ'.

Table 11: CPU supply R"'

	s_i	e_i
I_1	4	10
I_2	12	13
I_3	17	18
I_4	26	30

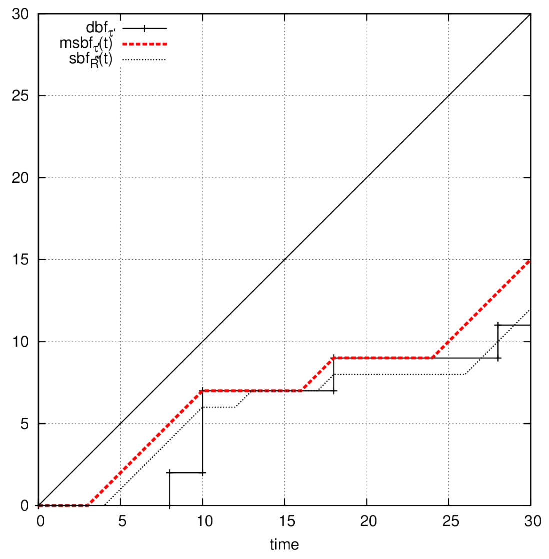


Figure 20: Graphical representation of $msbf_{\tau'}(t)$ and $sbf_{R'''}(t)$

As Figure 20 shows, if $sbf_{R'''}(t) \leq msbf_{\tau'}(t)$, in the minimum scheduling points, the CPU supply will not be enough to schedule the tasks. This is because at these points, $sbf_{R'''}(t) \leq dbf_{\tau'}(t)$.

All these counterexamples show that, in contrast to what we might think, the schedulability of a task set is not constrained by being in Zone 2. However, a task set will not be schedulable if its $sbf_R(t)$ is less than $msbf_\tau(t)$, that is, Zone 3.

As it has been demonstrated, the condition $sbf_R(t) \geq msbf_\tau(t)$ it is not enough because $sbf_R(t)$ can increase at the beginning and have a long idle interval and still be above $msbf_\tau(t)$ and miss deadlines during the long idle interval. However, a periodic resource $R = (\theta, \pi)$ ensures θ units of resource every π units of time. This, jointly with the condition $sbf_R(t) \geq msbf_\tau(t)$ ensures that the processor is assigned to τ whenever it is needed to not miss deadlines.

In an arbitrary CPU supply, it must be ensured that $sbf_R(t)$ provides enough CPU time *from time to time* but not necessarily periodically. With this idea, the objetive of the next section is to provide a schedulability test using $gsbf_\tau(t)$ and $msbf_\tau(t)$.

5. Schedulability analysis

Theorem 5 and Theorem 6 provide two alternative schedulability tests.

Let us use the definition of the $sbf_R(t)$ in definition 1 and $msbf_\tau(t)$ function in definition 7.

Theorem 5. *Let $sbf_R(t)$ be a function so that:*

$$msbf_\tau(t) \leq sbf_R(t) \leq t$$

and, moreover, $sbf_R(t)$ increases, at least, at the same time intervals as $msbf_\tau(t)$.

Then, τ is schedulable in R.

PROOF. It has been demonstrated that any sequence of slots $sbf_R(t)$ that coincides with $msbf_\tau(t)$ ensures the schedulability of $\{\tau, R\}$ (Section 3.2). So, if a sequence of slots supplies CPU time at the same time as $msbf_\tau(t)$ and also supplies it at another extra time, obviously, the task set will also be schedulable.

In other words, any function $sbf_R(t)$ that is above $msbf_\tau(t)$ and, moreover, increases, at least, at the same time intervals as $msbf_\tau(t)$, will be schedulable. Then, the gradient of $sbf_R(t)$ has to be equal to the gradient of $msbf_\tau(t)$ at least in $[s_j, e_j]$ to be schedulable, that is, both functions have to be parallel lines at least in that interval. See Figure 21.

The extra time mentioned previously represents CPU idle time.

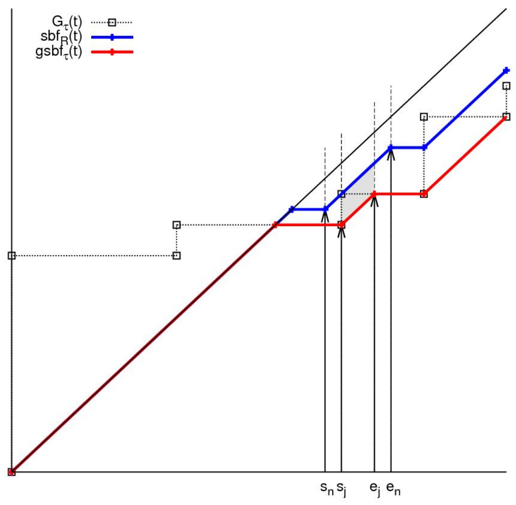


Figure 21: Conditions of schedulability in $\{\tau, R\}$

Lemma 4. *If $msbf_\tau(t) \leq sbf_R(t)$ and $sbf_R(t)$ increases, at least, at the same time intervals as $msbf_\tau(t)$, then the number of units of time that the CPU is idle are calculated as follows:*

$$CPU\_idle\_time = \sum_{\forall n} [e_n - s_n] - \sum_{\forall j} [e_j - s_j]$$

This situation can be illustrated by the following example. For the task set defined in Table 1, the $msbf_\tau(t)$ is obtained in Section 3.2.1 and the execution chronogram when $sbf_R(t) = msbf_\tau(t)$ is depicted in Figure 12.

Let us suppose another situation: an $sbf_R(t)$ that is greater than the previously calculated $msbf_\tau(t)$. Moreover, it also increases in the same intervals as $msbf_\tau(t)$ and also in other extra intervals, as shown in Figure 22.

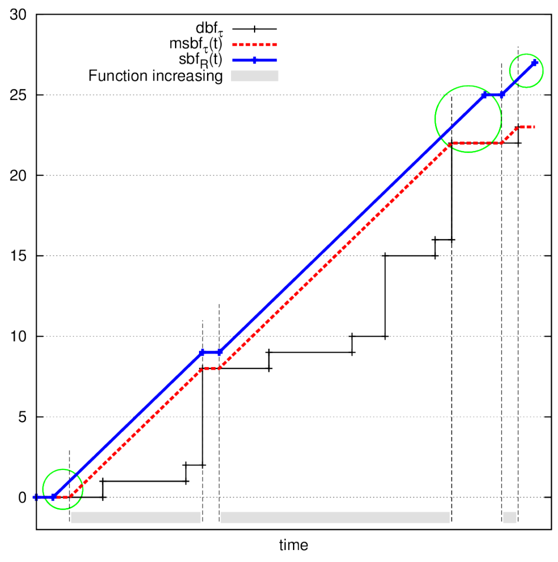


Figure 22: Example of schedulability in $\{\tau, R\}$

In this figure, $msbf_\tau(t)$ is depicted in red and $sbf_R(t)$ is depicted in blue. It is further observed that $sbf_R(t)$ increases at the same time as $msbf_\tau(t)$, that is, time intervals in grey. In addition, there are other time intervals in which $sbf_R(t)$ increases but $msbf_\tau(t)$ does not. They are highlighted in green.

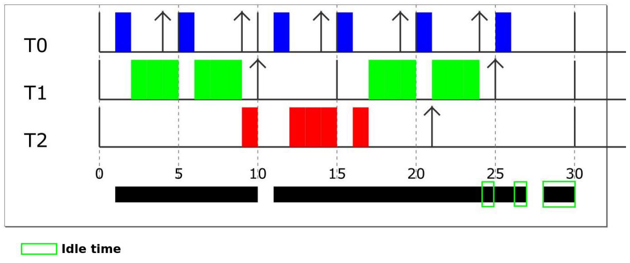


Figure 23: Chronogram execution of schedulability in $\{\tau, R\}$

Figure 23 shows the task set is schedulable in $\{\tau, R\}$ but there is CPU idle time. In fact, there is as much idle time as time instants when $sbf_R(t)$ increases but $msbf_\tau(t)$ does not.

So, the methodology for determining whether a function $sbf_R(t)$ ensures the schedulability of τ is:

1. Calculate the $msbf_\tau(t)$ for the task set, τ, and express it as set of intervals $I_j = [s_j, e_j]$ where the function increases.
2. Express $sbf_R(t)$ as a set of intervals $I_R = [s_R, e_R]$ in which the function increases.
3. Check if $I_j \subseteq I_R$, that is, $s_R \leq s_j \leq e_j \leq e_R, \forall j, R$.

The same happens with $gsbf_\tau(t)$. Let us use the definition of $gsbf_\tau(t)$ in definition 6.

Theorem 6. *Let $sbf_R(t)$ be a function so that:*

$$gsbf_\tau(t) \leq sbf_R(t) \leq t$$

and, moreover, $sbf_R(t)$ increases, at least, at the same time intervals in as $gsbf_\tau(t)$.

Then, τ is schedulable in R.

PROOF. In Section 3.1 it has been demonstrated that any sequence of intervals, R, defined by $sbf_R(t)$ that coincides with $gsbf_\tau(t)$ will ensure the schedulability of $\{\tau, R\}$. If, in addition to these intervals, other time instants are added to R, obviously τ will be also schedulable.

In other words: any $sbf_R(t)$ greater than $gsbf_\tau(t)$ and, whose gradient coincides with the gradient of $gsbf_\tau(t)$ in $[s_j, e_j]$ coincides. See Figure 24.

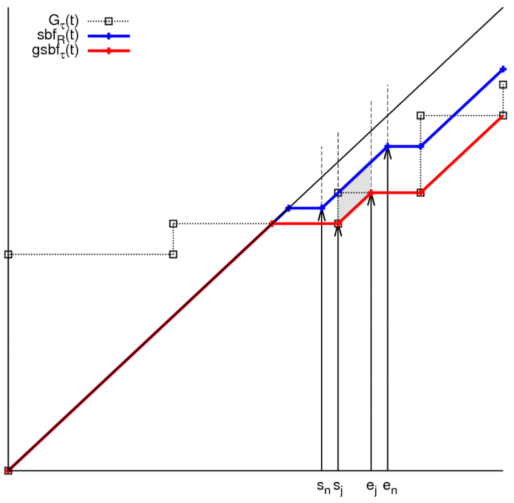


Figure 24: Conditions of schedulability in $\{\tau, R\}$

The additional time instants in which $sbf_R(t)$ increases but $gsbf_\tau(t)$ does not, represent the idle time of CPU.

Lemma 5. *If $gsbf_\tau(t) \leq sbf_R(t)$ and $sbf_R(t)$ increases, at least, at the same time intervals as $gsbf_\tau(t)$, then the number of units of time that the CPU is idle are calculated as follows:*

$$CPU\_idle\_time = \sum_{\forall n} [e_n - s_n] - \sum_{\forall j} [e_j - s_j]$$

This situation can be illustrated by the following example. For the task set defined in Table 1, the $gsbf_\tau(t)$ is obtained in Section 3.1.1 and the execution chronogram when $sbf_R(t) = gsbf_\tau(t)$ is depicted in Figure 8.

Let us suppose another situation: an $sbf_R(t)$ that is greater than the previously calculated $gsbf_\tau(t)$. Moreover, it also increases in the same intervals as $gsbf_\tau(t)$ and in other extra intervals, as shown in Figure 25.

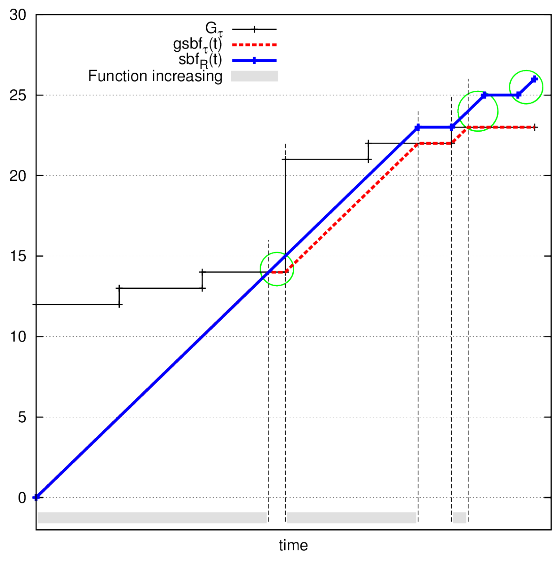


Figure 25: Example of schedulability in $\{\tau, R\}$

In this figure, $gsbf_\tau(t)$ is depicted in red and $sbf_R(t)$ is depicted in blue. It is further observed that the $sbf_R(t)$ increases at the same time as $gsbf_\tau(t)$, that is, time intervals in grey. In addition, there are other time intervals in which $sbf_R(t)$ increases but $gsbf_\tau(t)$ does not. They are highligthed in green.

As Figure 26 shows, the task set is schedulable in $\{\tau, R\}$ but there is CPU idle time. In fact, there is as much idle time as time instants when $sbf_R(t)$ increases but $gsbf_\tau(t)$ does not.

So, the methodology to be followed to determine if a function $sbf_R(t)$ ensures the schedulability of τ is the same as the case of $msbf_\tau(t)$:

1. Calculate the $gsbf_\tau(t)$ for task set, τ, and express it as set of intervals $I_j = [s_j, e_j]$ where the function increases.
2. Express $sbf_R(t)$ as a set of intervals $I_R = [s_R, e_R]$ in which the function increases.

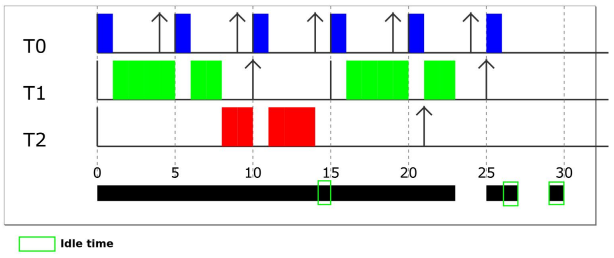


Figure 26: Chronogram execution of schedulability in $\{\tau, R\}$

3. Check if $I_j \subseteq I_R$, that is, $s_R \leq s_j \leq e_j \leq e_R, \forall j, R$.

In this section, a condition of schedulability has been proposed. In previous sections, two different schedulability algorithms have been presented ($msbf_\tau(t)$ and $gsbf_\tau(t)$) and, now, we have suggested a way of checking the schedulability of any sequence of slots by comparing to these functions.

It can be concluded that not all the $sbf_R(t)$ greater than $msbf_\tau(t)$ or $gsbf_\tau(t)$ will ensure the schedulability of $\{\tau, R\}$ but, in the specific above-mentioned conditions, it will.

6. Simulations

The main purpose of this section is to establish a relation between any $sbf_R(t)$ and the $msbf_\tau(t)$ or $gsbf_\tau(t)$, to ensure the schedulability of τ in R. What seems certain is that, if $sbf_R(t)$ is equal to or grows at the same time as $msbf_\tau(t)$ or $gsbf_\tau(t)$, τ will be schedulable in R, as proven in previous sections.

Our first hypothesis consisted in the fact that any $sbf_R(t)$ function greater than $msbf_\tau(t)$ or $gsbf_\tau(t)$ would guarantee the feasibility of τ in R. In Section 4, this theory has been rejected. Moreover, in that section, we differentiated the three zones in accordance with the position of $sbf_R(t)$ in relation to $msbf_\tau(t)$ and $gsbf_\tau(t)$ (Figure 13).

In this section, the percentage of schedulable task sets according to the zone where $sbf_R(t)$ is located is calculated. To this end, we simulate two situations:

- For the task set defined in Table 7, we generate different random sequences of slots, that is, $sbf_R(t)$, and we check the schedulability of the task set in R. The $sbf_R(t)$ generated is totally random so that the function generated can be contained in any of the zones mentioned before.

  After 300000 random sets generated, the results are as follows:

  - In cases where $sbf_R(t) < msbf_\tau(t)$, τ is never schedulable in R, as stated in Theorem 5.
  - In cases where $msbf_\tau(t) \leq sbf_R(t) < gsbf_\tau(t)$, the percentage of task set τ schedulable in R is 76.30%.

- In cases where $sbf_R(t) > gsbf_\tau(t)$, the percentage of task set τ schedulable in R is 72.45%.

On the basis of the results, we can deduce that although percentages are quite similar, the greater the $sbf_R(t)$, the more probability that τ will be non-schedulable. In other words, supplying more computation time than the time defined by $msbf_\tau(t)$ or $gsbf_\tau(t)$ does not imply that the task set is schedulable.

- We generate *n* task sets for each utilization factor U_t in $[0.4, 0.9]$, with $\triangle U_t = 0.1$. With $n = 300000$ iterations, the results are as follows:
  - In cases where $msbf_\tau(t) \leq sbf_R(t) < gsbf_\tau(t)$, the percentage of task set τ schedulable for each utilization factor is depicted in Figure 27.

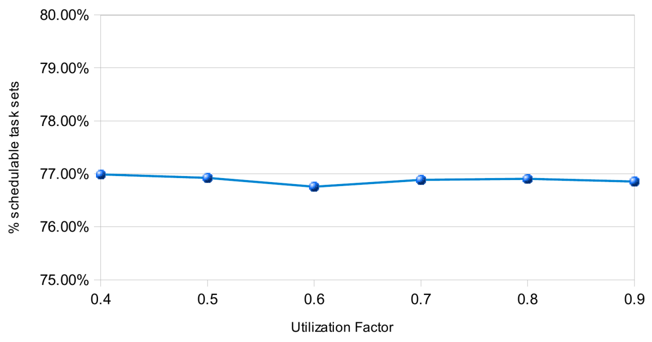


Figure 27: % schedulable τ in $msbf_\tau(t) \leq sbf_R(t) < gsbf_\tau(t)$

  - In cases where $sbf_R(t) > gsbf_\tau(t)$, the percentage of schedulable task set τ for each utilization factor is depicted in Figure 28.

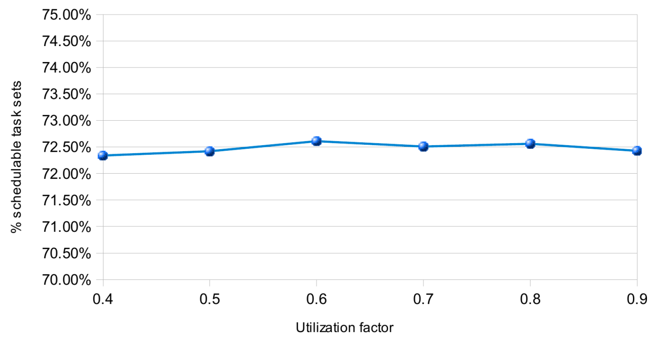


Figure 28: % schedulable τ in $sbf_R(t) > gsbf_\tau(t)$

According to the results, we can deduce that the utilisation factor is independent of the feasibility of the proposed method; that is, the fraction of time that the processor is busy does not contribute to a better or worse schedulability. And, moreover, as shown in the previous situation, the longer the $sbf_R(t)$, the more probability that τ will be non-schedulable (5% approximately).

7. Comparison of the minimum CPU supply function with other similar methods

As we stated in the introduction, there are no works related to the analysis of arbitrary global CPU. Nevertheless, we can compare our methods with other works that assume a periodic supply at this level. Specifically, [14] presents a schedulability analysis for a periodic supply (R=(θ, π)) at global level ($D_i = P_i \quad \forall i$) and EDF at local level with the following condition:

$$\forall 0 < t < lcm_\tau \quad dbf_\tau(t) \le sbf_R(t)$$

In this paper, to find out wheter a specific periodic supply R sucessfully schedules a set of tasks, the above equation must be simulated to obtain a solution space for θ and π. This equation was solved for the following set of tasks: $\tau = (T_1(7,50), T_2(9,75))$. When $\pi = 10$, the minimum θ was 2.8.

We are going to obtain the minimum θ using our $msbf_\tau(t)$. In the example, the characteristic points are $t_1 = 50$, $t_2 = 75$, $t_3 = 100$ and $t_4 = 150$. At these points the values of $msbf_\tau(t)$ are: $msbf_\tau(t_1) = 7$, $msbf_\tau(t_2) = 16$, $msbf_\tau(t_3) = 23$ and $msbf_\tau(t_4) = 39$.

We have to be sure that any periodic $sbf_R(t)$ with a period of 10 must be on top of $msbf_\tau(t)$ at the characteristic points. Therefore, at t_1, 7 units of time have at least been served. It is easy to see that to fulfill this condition, at t_1 then:

$$\theta = \frac{msbf_\tau(t_1)}{\frac{t_1}{\pi}} = \frac{7}{\frac{50}{10}} = 1.4.$$

Repeating this procedure for other scheduling points and choosing the maximum θ of all, then $\theta = 2.6$. This value is lower that the one calculated by Shin and Lee, which means that their proposal is an approximation and not an exact calculation. In fact, for $\theta = 2.5$ the task set is not schedulable. Figure 29 shows the comparison between $msbf_\tau(t)$ and the periodic supplies with the minimum θ calculated by Shin and Lee and by us.

8. Related work

In a partitioned architecture, partitions can be viewed as components that consist of a real-time workload and a scheduling policy for the workload. This definition coincides with a hierarchically organized compositional system. Different works in compositional scheduling have been proposed for a variety of real-time task models ([15, 16, 13, 5]).

Hierarchical scheduling has been a topic of research interest in recent years. One of the first works in this area is the one presented by Deng and Liu [17], based on a two-level real-time scheduling framework. Kuo and Li [18] presented an exact schedulability condition for this framework assuming fixed priority pre-emptive scheduling.

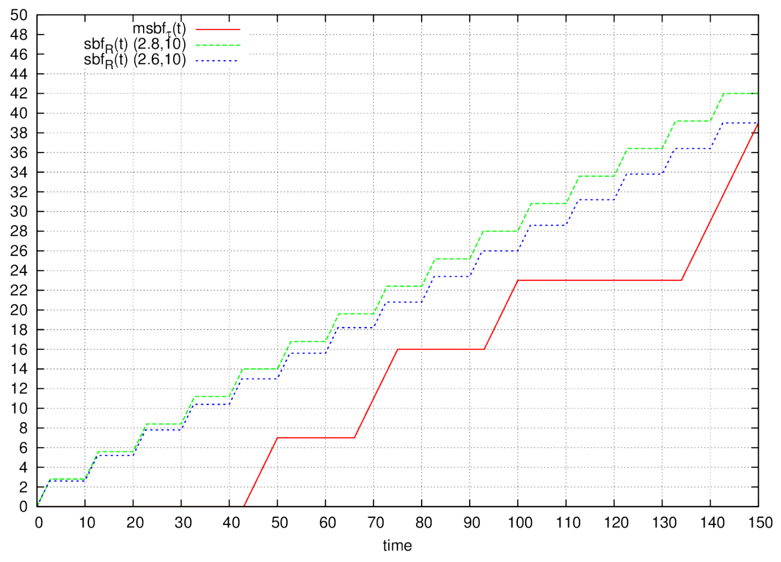


Figure 29: Comparison between two periodic supplies R=(2.6,10) and R=(2.8,10) and $msbf_\tau(t)$ for $\tau = (T_1(7, 50), T_2(9, 75))$

Lipari and Baruah [19] presented a similar work with EDF. Saewong et al. [20] provided a response time analysis for fixed-priority hierarchical systems. This analysis was pessimistic as shown by Davis and Burns in [21]. However, Davis and Burns gave an exact response time calculation only for the partition with the highest priority. Almeida and Pedreiras [22] improved the analysis by Saewong et al. but, again it was not an exact analysis. In [23], Bril et al. show that the worst case response time of a task is not necessarily assumed for the first job with the critical instant conditions provided by Davis and Burns. They claim that the existing analysis could be improved but they do not provide an alternative analysis. The exact response time was provided by Balbastre et al. in [24]. Lipari and Bini [25] provide a kind of sensitivity analysis of the global level for a two-level hierarchical system, providing a methodology for calculating the domain parameters that makes the task set feasible.

Lorente and Palencia [26] presented a worst-case response time analysis for the tasks in a two level hierarchical EDF systems. Zhang and Burns [27] propose an exact schedulability analysis for the application tasks when the local scheduler is EDF.

9. Conclusions

This article considers the case of a two level hierarchical real-time system, where local level tasks tasks are scheduled under EDF policy whereas the global level does not follow a known scheduling policy and the only information is provided as a set of CPU slots.

The first contribution is the calculation of CPU supply functions that ensure the schedulability of task sets; specifically, two different functions are defined: firstly, a

CPU supply which offers CPU when tasks are released ($gsbf$) and, secondly, a CPU supply which offers CPU as late as possible ($msbf$), meeting the deadlines in both situations and providing the minimum CPU.

This article also provides a schedulability analysis of a task set, relating any CPU supply with the functions mentioned before. In other words, the proposed method can be used to check if a set of time intervals provided by the SI can be accepted by the PD or not.

Moreover, our model can also be used with any existing server in the global level since it is a generalization of any scheduling in the global level. Although we initially apply our results to static scheduling, further work will focus on providing on-line algorithms to calculate $msbf_\tau(t)$ in a specific window. This could be especially useful in flexible environments, where even if the scheduling algorithm is known a priori, jitter or latencies makes final slots execution difficult to predict.

Improvements in the $msbf_\tau(t)$ and $gsbf_\tau(t)$ calculation are also foreseen. The proposed algorithm needs all the hyperperiod space to be exact. This value could be a large number so an upper bound for $msbf_\tau(t)$ and $gsbf_\tau(t)$ might be obtained to avoid studying the complete hyperperiod.

To conclude, we will also extend this study to consider fixed priorities in the local level.

[1] J. Windsor, K. Hjortnaes, Time and space partitioning in spacecraft avionics, in: IEEE Conference on Space Mission Challenges for Information Technology, 2009.

[2] IMA-SP Integrated Modular Avionics for Space. ESA project 4000100764 (2011-13).

[3] IMA-SP. Integrated Modular Avionics for Space, IMA Development Process, Roles and Tools, IMA-SP D08-11 (2011-13).

[4] Avionics Application Software Standard Interface (ARINC-653)., Airlines Electronic Eng. Committee (March 2006 2006).

[5] A. Easwaran, I. Lee, I. Shin, O. Sokolsky, Compositional schedulability analysis of hierarchical real-time systems, in: ISORC, 2007, pp. 274–281.

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

[7] S. Baruah, A. Mok, L. Rosier, Preemptively scheduling hard real-time sporadic tasks on one processor, in: IEEE Real-Time Systems Symposium, 1990, pp. 182–190.

[8] G. Buttazzo, in: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications., Kluwer Academic Publishers, 1997, pp. 110–115.

[9] S. Baruah, L. Rosier, R. Howell, Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor., in: The Journal of Real-Time Systems, 1990, Vol. 2, 1990, pp. 301–324.

[10] I. Ripoll, A. Crespo, A. Mok, Improvement in feasibility testing for real-time tasks, Journal of Real-Time Systems 11 (1996) 19–40.

[11] M. Spuri, Analysis of deadline schedule real-time systems, Technical Report 2772.

[12] V. Brocal, P. Balbastre, Selection of period for minimizing hyperperiod., RIAI - Revista Iberoamericana de Automatica e Informatica Industrial 10 (2) (2013) 197–203.

[13] I. Shin, I. Lee, Periodic resource model for compositional real-time guarantees, in: Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE, 2003, pp. 2–13. doi:10.1109/REAL.2003.1253249.

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

[15] S. Marimuthu, S. Chakraborty, A framework for compositional and hierarchical real-time scheduling, in: 12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2006, pp. 91 –96. doi:10.1109/RTCSA.2006.7.

[16] G. Lipari, E. Bini, Resource partitioning among real-time applications, in: Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on, 2003, pp. 151–158. doi:10.1109/EMRTS.2003.1212738.

[17] Z. Deng, J. W.-S. Liu, Scheduling real-time applications in an open environment, in: IEEE Real-Time Systems Symposium, 1997.

[18] T.-W. Kuo, C.-H. Li, A fixed priority driven open environment for real-time applications, in: IEEE Real-Time Systems Symposium, 1998.

[19] G. Lipari, S. Baruah, Efficient scheduling of real-time multi-task applications in dynamic systems, in: IEEE Real-Time Tecnology and Applications Symposium, 2000.

[20] S. Saewong, R. Rajkumar, J. Lehoczky, Analysis of hierarchical fixed-priority scheduling, in: Euromicro Conference on Real-Time Systems, 2002.

[21] R. I. Davis, A. Burns, Hierarchical fixed priority pre-emptive scheduling, in: IEEE Real-Time Systems Symposium, 2005.

[22] L. Almeida, P. Pedreiras, Scheduling within temporal partitions: response-time analysis and server design, in: Fourth ACM International Conference on Embedded Software (EMSOFT), 2004.

[23] R. J. Bril, W. F. J. Verhaegh, C. C. Wust, A cognac-glass algorithm for conditionally guaranteed budgets, in: IEEE Real-Time Systems Symposium, 2006.

[24] P. Balbastre, I. Ripoll, A. Crespo, Exact response time analysis of hierarchical fixed-priority scheduling, in: 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09., 2009, pp. 315–320. doi:10.1109/RTCSA.2009.40.

[25] G. Lipari, E. Bini, A methodology for designing hierarchical scheduling systems, Journal of Embedded Computing 1 (2) (2005) 257–269.

[26] J. Lorente, J. Palencia, An edf hierarchical scheduling model for bandwidth servers, in: Proceedings. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2006., 2006, pp. 261–266. doi:10.1109/RTCSA.2006.13.

[27] F. Zhang, A. Burns, Analysis of hierarchical edf pre-emptive scheduling, in: 28th IEEE International Real-Time Systems Symposium, 2007. RTSS 2007., 2007, pp. 423–434. doi:10.1109/RTSS.2007.12.