Overrun Methods for Semi-Independent Real-Time Hierarchical Scheduling

Moris Behnam, Thomas Nolte, Mikael Sjödin

Mälardalen Real-Time Research Centre

P.O. Box 883, SE-721 23 Västerås, Sweden

moris.behnam@mdh.se

Insik Shin

Dept. of Computer Science, KAIST

Daejeon, South Korea 305-701

insik.shin@cs.kaist.ac.kr

Abstract

The Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework to enable compositional schedulability analysis of embedded software systems with real-time properties. In this paper a software system consists of a number of semi-independent components called subsystems. Subsystems are developed independently and later integrated to form a system. To support this design process, in the paper, the proposed methods allow non-intrusive configuration and tuning of subsystem timing-behaviour via subsystem interfaces for selecting scheduling parameters.

This paper considers three methods to handle overruns due to resource sharing between subsystems in the HSF. For each one of these three overrun methods corresponding scheduling algorithms

The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

and associated schedulability analysis are presented together with analysis that shows under what circumstances one or the other is preferred. The analysis is generalized to allow for both Fixed Priority Scheduling (FPS) and Earliest Deadline First (EDF) scheduling. Also, a further contribution of the paper is the technique of calculating resource-holding times within the framework under different scheduling algorithms. The resource holding times being an important parameter in the global schedulability analysis.

I. Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical resource sharing among applications under different scheduling services. The hierarchical scheduling framework can be generally represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., threads), and resources are allocated from a parent node to its children nodes.

The HSF provides means for decomposing a complex system into well-defined parts. In essence, the HSF provides a mechanism for timing-predictable *composition* of course-grained components or *subsystems*. In the HSF a subsystem provides an introspective *interface* that specifies the timing properties of the subsystem precisely [28]. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal behaviour. Also, the HSF facilitates *reusability* of subsystems in timing-critical and resource constrained environments, since the well defined interfaces characterize their computational requirements.

Earlier efforts have been made in supporting compositional subsystem integration in the HSFs, preserving the independently analyzed schedulability of individual subsystems. One common assumption shared by earlier studies is that subsystems are independent. This paper relaxes this assumption by addressing the challenge of enabling efficient compositional integration for independently developed *semi-independent* subsystems interacting through sharing of mutual exclusion access logical resources. Here, semi-independence means that subsystems are allowed to synchronize by the sharing of logical resources.

To enable sharing of logical resources in HSFs, Davis and Burns proposed a synchronization protocol implementing the *overrun* mechanism, allowing the subsystem to overrun (its budget) to complete the execution of a critical section [11]. Two versions of overrun mechanisms were presented in [11], called overrun without payback and overrun with payback, and in the remainder of this paper these overrun mechanisms are called Basic Overrun (BO), and Basic Overrun with Payback (PO), respectively. The study presented by Davis and Burns provides schedulability analysis for both overrun mechanisms; however, the schedulability analysis does not allow independent analysis of individual subsystems. Hence, the presented schedulability analysis does not naturally support composability of subsystems.

The schedulability analysis of Davis and Burns' has been extended assessing composability in [8] for systems running the Earlier Deadline First (EDF) scheduling algorithm. In addition, in the same paper a new overrun mechanism has been presented, called Enhanced Overrun (EO), that potentially increases schedulability within a subsystem by providing CPU allocations more efficiently. Also, in the paper this new mechanism has been evaluated against PO.

The contributions of this paper are as follows; Firstly, BO, the second version of overrun mechanism presented in [11], is included in the comparison between overrun mechanisms presented in [8] and it is shown under which circumstances where a certain overrun mechanism is the preferred one among all three (BO, PO and EO) presented mechanisms. In addition, the sechedulability analysis of local and global schedulers is generalized by including Fixed Priority Scheduling (FPS) in the schedulability analysis, as the results of [8] were limited to the EDF scheduling algorithm. Finally, the simplified equation to calculate resource holding time using the EDF scheduling algorithm (presented in [8]) is proven to be valid also when using the FPS scheduling algorithm. Hence, using the results of this paper it is possible to use either FPS or EDF.

The outline of the paper is as follows: Section II presents related work, while Section III presents the system model. In Section IV the schedulability analysis for the system model is presented. Section V presents the three overrun mechanisms (BO, PO and EO), and Section VI presents their analytical comparison. In Section VII it is shown how to calculate the resource holding times under both FPS and EDF, and finally, Section VIII concludes.

II. Related work

This section presents related work in the areas of HSFs as well as resource sharing protocols.

A. Hierarchical scheduling

The HSF for real-time systems, originating in open systems [12] in the late 1990's, has been receiving an increasing research attention. Since Deng and Liu [12] introduced a two-level HSF, its schedulability has been analyzed under fixed-priority global scheduling [17] and under EDF-based global scheduling [19], [22]. Mok *et al.* [24] proposed the bounded-delay resource model so as to achieve a clean separation in a multi-level HSF, and schedulability analysis techniques [14], [29] have been introduced for this resource model. In addition, Shin and Lee [28], [30] introduced another periodic resource model (to characterize the periodic resource allocation behaviour), and many studies have been proposed on schedulability analysis with this resource model under fixed-priority scheduling [26], [20], [10] and under EDF scheduling [28]. More recently, Easwaran *et al.* [13] introduced Explicit Deadline Periodic (EDP) resource model. However, a common assumption shared by all the studies in this paragraph is that tasks are required to be independent.

B. Resource sharing

In many real systems, tasks are semi-independent, interacting with each other through mutually exclusive resource sharing. Many protocols have been introduced to address the priority inversion problem for semi-independent tasks, including the Priority Inheritance Protocol (PIP) [27], the Priority Ceiling Protocol (PCP) [25], and Stack Resource Policy (SRP) [3]. Recently, Fisher *et al.* addressed the problem of minimizing the resource holding time [16] under SRP. There have been studies on extending SRP for HSFs, for sharing of logical resources within a subsystem [2], [17] and across subsystems [11], [7], [15]. Davis and Burns [11] proposed the Hierarchical Stack Resource Policy (HSRP) supporting sharing of logical resources on the basis of an overrun mechanism. Behnam *et al.* [7] proposed the Subsystem Integration and Resource Allocation Policy (SIRAP) protocol that supports subsystem integration in the presence of shared logical resources, on the basis of skipping. Fisher *et al.* [15] proposed the BROE server that extends the Constant Bandwidth Server (CBS) [1] in order to handle sharing of logical resources in a HSF. Behnam*et al.* [6] compared between SIRAP, HSRP and BROE and showed that there is no one silver bullet solution available today, providing an optimal HSF and synchronization protocol for use in open environments. Lipari *et al.* proposed the BandWidth Inheritance protocol (BWI) [23] which extends the resource reservation framework to systems where tasks can share resources. The BWI approach is based on using the CBS algorithm and

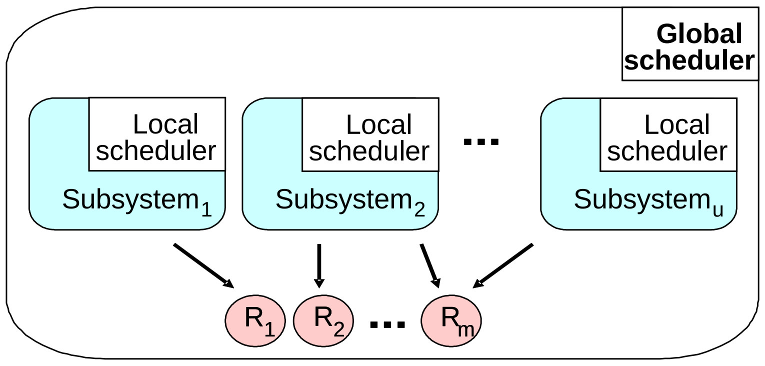

Fig. 1. Two-level HSF with resource sharing.

a technique that is derived from the Priority Inheritance Protocol (PIP). Particularly, BWI is suitable for systems where the execution time of a task inside a critical section can not be evaluated.

III. System model and background

A. Resource sharing in the HSF

The Hierarchical Scheduling Framework (HSF) has been introduced to support CPU time sharing among applications (subsystems) under different scheduling policies. In this paper, a two level-hierarchical scheduling framework is considered, which works as follows: a global (system-level) scheduler allocates CPU time to subsystems, and a local (subsystem-level) scheduler subsequently allocates CPU time to its internal tasks.

Having such a HSF also allows for the sharing of logical resources among tasks in a mutually exclusive manner (see Figure 1). Specifically, tasks can share *local* logical resources within a subsystem as well as *global* logical resources across (in-between) subsystems. However, note that this paper focuses around mechanisms for sharing of global logical resources in a HSF while local logical resources easily can be supported by traditional synchronization protocols such as SRP (see, e.g., [2], [11], [17]).

B. Virtual processor models

The notion of real-time virtual processor (resource) model was first introduced by Mok *et al.* [24] to characterize the CPU allocations that a parent node provides to a child node in a HSF. The *CPU* *supply* of a virtual processor model refers to the amounts of CPU allocations that the virtual processor model can provide. The *supply bound function* of a virtual processor model calculates its minimum possible CPU supply for any given time interval of length t.

The periodic virtual processor model $\Gamma(P, Q)$ was proposed by Shin and Lee [28] to characterize periodic resource allocations, where P is a period ($P > 0$) and Q is a periodic allocation time ($0 < Q \leq P$). The capacity U_Γ of a periodic virtual processor model $\Gamma(P, Q)$ is defined as Q/P.

The *supply bound function* $\texttt{sbf}_\Gamma(t)$ of the periodic virtual processor model $\Gamma(P, Q)$ was given in [28] to compute the minimum resource supply during an interval of length t. Further, in this paper, the periodic virtual processor model is rephrased with an additional parameter of BD, where BD represents its longest possible *blackout duration* during which the periodic virtual processor model may provide no resource allocation at all.

$$\texttt{sbf}_\Gamma(t, \mathsf{BD}) = \begin{cases} t - (k-1)(P-Q) - BD & \text{if } t \in W^{(k)} \\ (k-1)Q & \text{otherwise,} \end{cases} \tag{1}$$

where $k = \max\Big(\lceil (t + (P-Q) - BD)/P \rceil, 1\Big)$ and $W^{(k)}$ denotes an interval $[(k-1)P + \mathsf{BD}, (k-1)P + \mathsf{BD} + Q]$. Here, first note that the original $\texttt{sbf}_\Gamma(t)$ in [28] is equivalent to $\texttt{sbf}_\Gamma(t, \mathsf{BD})$ when $\mathsf{BD} = 2(P-Q)$. Also, note that an interval of length t may not begin synchronously with the beginning of period P; as shown in Figure 2, the interval of length t can start in the middle of the period of a periodic virtual processor model $\Gamma(P, Q)$. Figure 2 illustrates the supply bound function $\texttt{sbf}_\Gamma(t)$ of the periodic virtual processor model.

C. Stack resource policy (SRP)

To be able to use SRP [3] in the HSF, its associated terms are extended as follows:

- *Preemption level*. Each task τ_i has a preemption level equal to $\pi_i = 1/D_i$, where D_i is the relative deadline of the task. Similarly, each subsystem S_s has an associated preemption level equal to $\Pi_s = 1/P_s$, where P_s is the subsystem's per-period deadline.
- *Resource ceiling*. Each globally shared resource R_j is associated with two types of resource ceilings; one *internal* resource ceiling for local scheduling $rc_j = \max\{\pi_i | \tau_i \text{ accesses } R_j\}$ and one *external* resource ceiling for global scheduling.
- *System/subsystem ceilings*. System/subsystem ceilings are dynamic parameters that change during runtime. The system/subsystem ceiling is equal to the currently locked highest external/internal

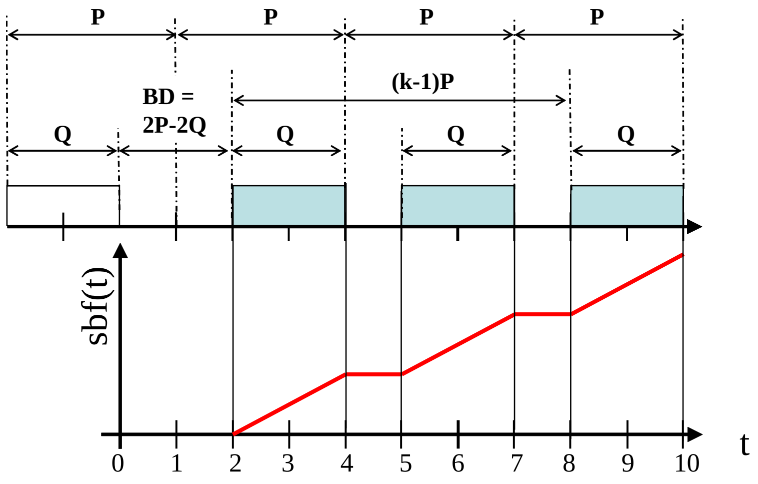

Fig. 2. The supply bound function of a periodic virtual processor model $\Gamma(3, 2)$.

resource ceiling in the system/subsystem.

Following the rules of SRP, a job J_i that is generated by a task τ_i can preempt the currently executing job J_k within a subsystem only if J_i has a priority higher than that of job J_k and, at the same time, the preemption level of τ_i is greater than the current subsystem ceiling. A similar reasoning is made for subsystems from a global scheduling point of view.

D. System model

In this paper a periodic task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$ is considered, where T_i, C_i and D_i represent the task's period, worst-case execution time (WCET) and relative deadline, respectively, where $D_i \leq T_i$, and $\{c_{i,j}\}$ is the set of WCETs within critical sections associated with task τ_i. Each element $c_{i,j}$ in $\{c_{i,j}\}$ represents the WCET of the task τ_i inside a critical section of the global shared resource R_j.

Looking at a shared resource R_j, the *resource holding time* $h_{j,i}$ of a task τ_i is defined as the time given by the task's maximum execution time inside a critical section plus the interference (inside the critical section) of higher priority tasks having preemption level greater than the internal ceiling of the locked resource.

A subsystem $S_s \in \mathcal{S}$, where $\mathcal{S}$ is the whole system of subsystems, is characterized by a task set $\mathcal{T}_s$ and a set of internal resource ceilings $\mathcal{R}C_s$ inherent from internal tasks using the globally shared resources. Each subsystem S_s is assumed to have an EDF or FPS local scheduler, and the subsystems are scheduled according to EDF or FPS on a global level. The collective resource requirements by each subsystem S_s is characterized by its *interface* (the subsystem interface) defined as (P_s, Q_s, H_s), where P_s is the subsystem's period, Q_s is it's execution requirement budget, and H_s is the subsystem's maximum resource holding time, i.e., $H_s = \max\{h_{j,i}|\tau_i \in \mathcal{T}_s \text{ accesses } R_j\}$.

IV. Schedulability analysis

This section presents the schedulability analysis of the HSF, starting with local schedulability analysis needed to calculate subsystem interfaces, and finally, global schedulability analysis. The analysis presented assumes that SRP is used for synchronization on the local (within subsystems) level.

A. Local schedulability analysis

Let $\mathtt{dbf}_{\mathsf{EDF}}(i,t)$ denote the demand bound function of a task τ_i under EDF scheduling [4], i.e.,

$$\mathtt{dbf}_{\mathsf{EDF}}(i,t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \cdot C_i. \tag{2}$$

The local schedulability condition under EDF scheduling is then (by combining the results of [5] and [28])

$$\forall t > 0 \quad \sum_{i=1}^{n} \mathtt{dbf}_{\mathsf{EDF}}(i,t) + b(t) \leq \mathtt{sbf}(t), \tag{3}$$

where $b(t)$ is the blocking function [5] that represents the longest blocking time during which a job J_i with $D_i \leq t$ may be blocked by a job J_k with $D_k > t$ when both jobs access the same resource.

For Fixed Priority Scheduling (FPS) [18], let $\mathtt{rbf}_{\mathsf{FP}}(i,t)$ denote the request bound function of a task τ_i, i.e.,

$$\mathtt{rbf}_{\mathsf{FP}}(i,t) = C_i + \sum_{\tau_k \in \mathtt{HP}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \tag{4}$$

where $\mathtt{HP}(i)$ is the set of tasks with higher priorities than that of τ_i. The local schedulability analysis under FPS can then easily be extended from the results of [3], [28] as follows:

$$\forall \tau_i, 0 < \exists t \leq D_i, \ \mathtt{rbf}_{\mathsf{FP}}(i,t) + b_i \leq \mathtt{sbf}(t), \tag{5}$$

where b_i is the maximum *blocking* (i.e., extra CPU demand) imposed to a task τ_i when τ_i is blocked by lower priority tasks that are accessing resources with ceiling greater than or equal to the priority of τ_i. Note that t can be selected within a finite set of scheduling points [21].

B. Subsystem interface calculation

Given a subsystem S_s, $\mathcal{RC}_s$, and P_s, let $\mathsf{calculateBudget}(S_s, P_s, \mathcal{RC}_s)$ denote a function that calculates the smallest subsystem budget Q_s that satisfies Eq. (3) for EDF and Eq. (5) for FPS scheduling. Hence, $Q_s = \mathsf{calculateBudget}(S_s, P_s, \mathcal{RC}_s)$. The function is similar to the one presented in [28], however, due to space limitations, its details are left out of this paper.

C. Global schedulability analysis

Following Theorem 1 of [5], global schedulability analysis under EDF scheduling is given using the system load bound function $\mathtt{LBF}(t)$ as follows:

$$\forall t > 0, \quad \mathtt{LBF}(t) = B(t) + \sum_{S_s \in \mathcal{S}} \mathtt{DBF}_s(t) \leq t, \tag{6}$$

where

$$\mathtt{DBF}_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \cdot Q_s, \tag{7}$$

and the system-level blocking function $B(t)$ represents the maximum blocking time during which a subsystem S_s may be blocked by another subsystem S_k, where $P_s \leq t$ and $P_k > t$. $B(t)$ is defined as

$$B(t) = \max\{H_k \mid P_k > t\}. \tag{8}$$

Under global FPS scheduling, the subsystem load bound function is as follows (on the basis of a similar reasoning of Eq. (4)):

$$\mathtt{LBF}_s(t) = \mathtt{RBF}_s(t) + B_s \text{ , where} \tag{9}$$

$$\mathtt{RBF}_s(t) = Q_s + \sum_{S_k \in \mathtt{HPS}(s)} \left\lceil \frac{t}{P_k} \right\rceil Q_k, \tag{10}$$

where $\texttt{HPS}(s) = \{S_j | j > s\}$ is the set of subsystems with priority higher than that of S_s. Let B_s denote the maximum blocking (i.e., extra CPU demand) imposed to a subsystem S_s, when it is blocked by lower-priority subsystems,

$$B_s = \max\{H_j | \ S_j \in \texttt{LPS}(S_s)\}, \tag{11}$$

where $\texttt{LPS}(S_s) = \{S_j | j < s\}$.

A global schedulability condition under FPS is then

$$\forall S_s, 0 < \exists t \leq P_s, \ \texttt{LBF}_s(t) \leq \texttt{t} \tag{12}$$

V. Overrun mechanisms

This section explains three overrun mechanisms that can be used to handle budget expiry during a critical section in the HSF. Consider a global scheduler that schedules subsystems according to their periodic interfaces (P_s, Q_s, H_s). The subsystem budget Q_s is said to *expire* at the point when one or more internal (to the subsystem) tasks have executed a total of Q_s time units within the subsystem period P_s. Once the budget is expired, no new task within the same subsystem can initiate its execution until the subsystem's budget is replenished. This replenishment takes place in the beginning of each subsystem period, where the budget is replenished to a value of Q_s.

Budget expiration may cause a problem if it happens while a job J_i of a subsystem S_s is executing within a critical section of a global shared resource R_j. If another job J_k, belonging to another subsystem, is waiting for the same resource R_j, this job must wait until S_s is replenished again so J_i can continue to execute and finally release the lock on resource R_j. This waiting time exposed to J_k can be potentially very long, causing J_k to miss its deadline.

In this paper, an overrun mechanism is considered as follows; when the budget of subsystem S_s expires and S_s has a job J_i that is still locking a globally shared resource, job J_i continues its execution until it releases the locked resource. The extra time that J_i needs to execute after the budget of S_s expires is denoted as *overrun time* θ. The maximum θ occurs when J_i locks a resource that gives the longest resource holding time just before the budget of S_s expires.

Here, two versions of overrun mechanisms [11] are considered;

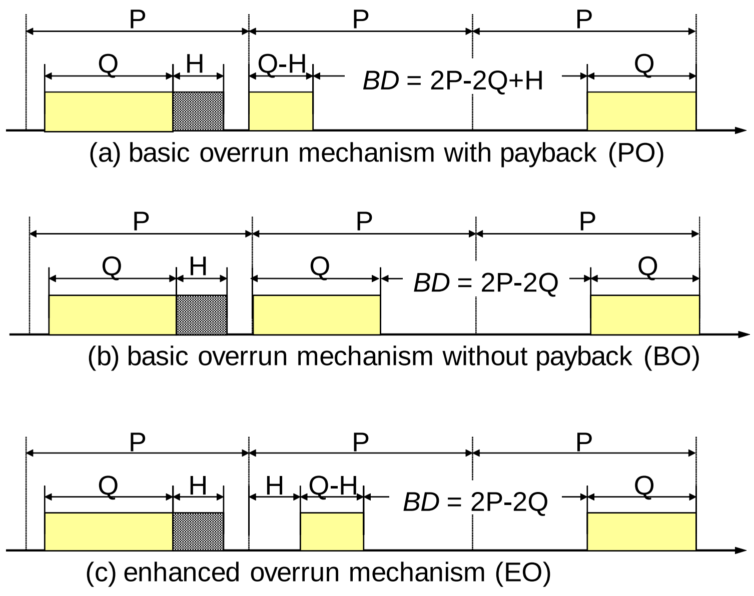
P
P
P
Q
H
Q-H
BD = 2P-2Q+H
Q
(a) basic overrun mechanism with payback (PO)
P
P
P
Q
H
Q
BD = 2P-2Q
Q
(b) basic overrun mechanism without payback (BO)
P
P
P
Q
H
H
Q-H
BD = 2P-2Q
Q
(c) enhanced overrun mechanism (EO)

Fig. 3. Basic and enhanced overrun mechanisms.

1) The overrun mechanism with payback, introduced as PO and later EO, whenever overrun happens, the subsystem S_s pays back θ in its next execution instant, i.e., the subsystem budget Q_s will be decreased by θ for the subsystem's execution instant following the overrun (note that only the instant following the overrun is affected).
2) The overrun mechanism without payback, introduced as BO, in this version of the overrun mechanism, no further actions will be taken after the event of an overrun.

Hereinafter, the overrun mechanism with payback is called PO, and the overrun mechanism without payback is called BO. Both are versions of the basic overrun mechanism. Also, an extended mechanism with payback is introduced as EO.

A. Basic overrun – overrun mechanism 1 and 2

Davis *et al.* [11] presented schedulability analysis for both (BO and PO) versions of basic overrun, however, the presented analysis is not suitable for open environments [12] as it requires detailed information of all tasks in the system in order to calculate global schedulability. This section discusses how to extend the existing schedulability analysis for the basic overrun mechanisms, making them suitable for open environments.

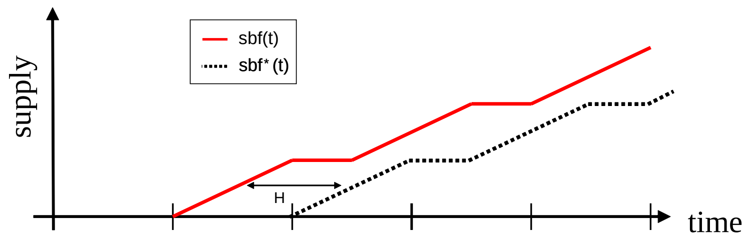

Fig. 4. Comparing $\mathtt{sbf}(\mathtt{t})$ **with** $\mathtt{sbf}^{\circ}(\mathtt{t})$**.**

1) Independent analysis with basic overrun: The supply bound function in [28] was developed under the assumption that the greatest blackout duration is $2(P-Q)$. PO – basic overrun with payback – cannot employ this existing supply bound function for schedulability analysis because its greatest Blackout Duration (BD) is $2(P-Q)+H$ (as shown in Figure 3a). Taking this into account, below is the presentation of a modified supply bound function $\mathtt{sbf}^{\circ}_{\Gamma}(t)$, that can be used with PO (using Eq. (1)), as follows:

$$\mathtt{sbf}^{\circ}_{\Gamma}(t) = \mathtt{sbf}_{\Gamma}(t, \mathsf{BD}^{\circ}), \text{ where } \mathsf{BD}^{\circ} = 2(P-Q)+H. \tag{13}$$

The existing schedulability conditions of Eq. (3) can then be extended by substituting $\mathtt{sbf}_{\Gamma}(t)$ with $\mathtt{sbf}^{\circ}_{\Gamma}(t)$.

For BO – basic overrun without payback – Eq. (1) can still be used without modification to evaluate the supply bound function since the Blackout Duration is $2(P-Q)$ (as shown in Figure 3b).

2) Global analysis with basic overrun:

a) PO – basic overrun with payback.: Firstly, the demand bound function (and the request bound function) of a subsystem with the basic overrun mechanism with payback is extended. Looking at the PO mechanism in a subsystem S_s, the maximum contribution on $\mathtt{DBF}_s(t)$ for EDF scheduling and $\mathtt{RBF}_s(t)$ for FPS scheduling is H_s. When S_s overruns with its maximum, which is H_s, the subsystem's resource demand within the subsystem period P_s will be increased to Q_s+H_s. Following this, the budget of the next period will be decreased to Q_s-H_s due to the payback mechanism. Then, suppose that the subsystem overruns again. Now, during the next subsystem period, the subsystem's resource demand will be $Q_s-H_s+H_s=Q_s$. Here, it is easy to observe that the subsystem's resource demand will be at most kQ_s+H_s during k subsystem periods. Hence, the demand bound function $\mathtt{DBF}^{\circ}_s(t)$ of a subsystem S_s with the basic overrun mechanism using EDF scheduling globally is

$$\mathtt{DBF}^{\circ}_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \cdot Q^{\circ}_s + O_s(t), \tag{14}$$

where Q°_s is the subsystem budget when using the PO mechanism and,

$$O_s(t) = \begin{cases} H_s & \text{if } t \geq P_s, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

The schedulability condition of Eq. (6) can then be extended by substituting $\mathtt{DBF}_s(t)$ with $\mathtt{DBF}^{\circ}_s(t)$.

When using a global FPS scheduler, the request bound function $\mathtt{RBF}^{\circ}_s(t)$ is

$$\mathtt{RBF}^{\circ}_s(t) = (Q^{\circ}_s + H_s) + \sum_{S_k \in \mathtt{HPS}(s)} (\left\lceil \frac{t}{P_k} \right\rceil (Q^{\circ}_k) + H_k) \tag{16}$$

b) BO – basic overrun without payback.: This version of overrun does not payback the budget after overrun happens. This means that the system resource demands within the period of P_s can be up to $Q_s + H_s$ for all periods considering that the maximum overrun will happen every period, which is the worst case scenario. Then for EDF global scheduling, the maximum demand bound function $\mathtt{DBF}^{\#}_s(t)$ using the BO mechanism is

$$\mathtt{DBF}^{\#}_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \cdot (Q^{\#}_s + H_s) \tag{17}$$

where $Q^{\#}_s$ is the subsystem budget when using the BO mechanism.

For a global FPS scheduler, the request bound function $\mathtt{RBF}^{\#}_s(t)$ is

$$\mathtt{RBF}^{\#}_s(t) = (Q^{\#}_s + H_s) + \sum_{S_k \in \mathtt{HPS}(s)} \left\lceil \frac{t}{P_k} \right\rceil (Q^{\#}_k + H_k) \tag{18}$$

B. Enhanced overrun – overrun mechanism 3

As seen in Section V-A, the PO mechanism works with a modified supply bound function $\mathtt{sbf}^{\circ}(t)$ that is less efficient in terms of CPU resource usage compared with the original sbf(t), as illustrated in Figure 4. While for the BO mechanism, the request/demand bound function (DBF/RBF) will be increased by $Q_s + H_s$ in all periods which may require more resources as well. In the following an enhanced overrun mechanism (EO) is proposed. This new overrun mechanism makes it possible to use sbf(t) in Eq. (1) to improve the efficiency of CPU resource utilization and at the same time the request/demand bound function ($\mathtt{DBF}/\mathtt{RBF}$) will be $Q_s + H_s$ for the first instance and then only Q_s for the following periods when applying global schedulability analysis.

The EO mechanism is based on imposing an offset (delaying the budget replenishment of subsystem) equal to the amount of an overrun θ_s to the execution instant that follows a subsystem overrun. As shown in Figure 3c, the execution of the subsystem will be delayed by θ_s after a new period followed by overrun even if that subsystem has the highest priority at that time. By this the maximum BD will be decreased to $2(P - Q)$ compared with PO (basic overrun with payback) shown in Figure 3a and therefore it is possible to use the same supply bound function presented in Section III-B. One of the important features that the EO mechanism provides is that it moves the effect of overrun from the local to the global schedulability analysis, so the subsystem development will not depend on if there is a specific overrun mechanism enforced or not. This feature is very important in an open environment, and it allows for the usage of the existing local schedulability condition without any modification.

1) Global analysis with enhanced overrun: The effect of overrun is now moved to global schedulability analysis when using the EO mechanism. In the following, a demand bound function $\mathtt{DBF}^*_s(t)$ is presented for EDF global scheduling of a subsystem S_s that upper-bounds the demand requested by S_s under the EO mechanism. Now, $\mathtt{DBF}^*_s(t)$ includes the offset $\theta_s = H_s$ as follows:

$$\mathtt{DBF}^*_s(t) = \left\lfloor \frac{t + H_s}{P_s} \right\rfloor \cdot Q^*_s + O^*_s(t), \tag{19}$$

where Q^*_s is the subsystem budget when using the EO mechanism and

$$O^*_s(t) = \begin{cases} H_s & \text{if } t \geq P_s - H_s, \\ 0 & \text{otherwise}. \end{cases} \tag{20}$$

The schedulability condition of Eq. (6) can then be extended by substituting $\mathtt{DBF}_s(t)$ with $\mathtt{DBF}^*_s(t)$.

Using an FPS global scheduler, the offset imposed by the EO mechanism for each subsystem S_s can be modeled as a release jitter J_s with the range of $[0, H_s]$ so $J_s = H_s$. The upper bound of request bound function $\mathtt{RBF}^*_s(t)$ calculation is shown below,

$$\mathtt{RBF}^*_s(t) = (Q^*_s + H_s) + \sum_{S_k \in \mathtt{HPS}(s)} (\left\lceil \frac{t + J_k}{P_k} \right\rceil (Q^*_k) + H_k) \tag{21}$$

Looking at the schedulability analysis then

$$\forall S_s, 0 < \exists t \le P_s - H_s,\ \mathtt{LBF}^*_s(t) \le \mathtt{t} \tag{22}$$

where

$$\mathtt{LBF}^*_s(t) = \mathtt{RBF}^*_s(t) + B_s\ ,\ \text{where} \tag{23}$$

VI. Comparison between the three overrun mechanisms

In this section, the efficiency of the three overrun mechanisms (BO, PO and EO) are compared. First, the effect of using each one of them locally is shown, i.e., on a subsystem level. Then, their effect globally is shown, i.e., on a system level.

A. Subsystem-level comparison

The following lemma shows that the minimum required subsystem budget when using the EO mechanism will be lower than or equal to the minimum required budget when using the PO mechanism for both FPS and EDF local schedulers.

Lemma 1: Assuming that the minimum required budget to schedule all tasks in a subsystem S_s using the PO mechanism is Q°_s, and that the corresponding budget when using the EO mechanism is Q^*_s, then $Q^*_s \le Q^\circ_s$.

Proof: The proof is split into two parts, proving the case of having an EDF local scheduler and an FPS local scheduler, respectively.

a) EDF local scheduler.: A subsystem S_s is exactly schedulable iff in addition to Eq. (3), $\sum_i^n \mathtt{dbf}_{\mathsf{EDF}}(i,t) + b(t) = \mathtt{sbf}(t)$ for $\exists t$ s.t. $min_i^n D_i \le t \le LCM_{S_s} + max_i^n D_i$ (see Theorem 2.2 in [13]). This means that if the budget Q_s is the minimum required budget to guarantee the schedulability of tasks in S_s, then there is a set of times t^e at which $\sum_i^n \mathtt{dbf}_{\mathsf{EDF}}(i,t) + b(t) = \mathtt{sbf}(t)$. Without loss of generality, assume that t^e includes one element. If the same subsystem budget Q_s is used when running the PO mechanism and the EO mechanism, respectively, then

$$\mathtt{sbf}^\circ(t) = \mathtt{sbf}(t - H_s) \tag{24}$$

where $\mathtt{sbf}(t)$ is used with the EO mechanism and the shift in time "$-H_s$" comes from the difference in BD between the EO and PO mechanisms. From Eq. (1) and Eq. (24), there are two cases:

case 1: $\mathtt{sbf}^{\circ}(t) = \mathtt{sbf}(t)$ for $t \in [kP_s - Q_s + H_s, (k+1)P_s - 2Q_s]$ where k is an integer number $k > 1$.
case 2: $\mathtt{sbf}^{\circ}(t) < \mathtt{sbf}(t)$ for t out of the range specified in case 1.

If $t^e \in [kP_s - Q_s + H_s, (k+1)P_s - 2Q_s]$ then $\mathtt{sbf}^{\circ}(t^e) = \mathtt{sbf}(t^e)$. In turn, $\sum_i^n \mathtt{dbf}_{\mathsf{EDF}}(i, t^e) + b(t^e) = \mathtt{sbf}^{\circ}(t^e)$, which means that Q_s may be enough to schedule all tasks in a subsystem S_s using the PO mechanism, so $Q_s^* = Q_s^{\circ}$ at time $t = t^e$. However, Eq. (3) must be checked if it holds for all other times t, to be sure that the subsystem S_s is still schedulable.

If t^e is not in the range given for case 1, then $\mathtt{sbf}^{\circ}(t^e) < \mathtt{sbf}(t^e)$. In turn, $\mathtt{sbf}^{\circ}(t^e) < \sum_i^n \mathtt{dbf}_{\mathsf{EDF}}(i, t^e) + b(t^e)$ which means that the budget Q_s will not satisfy the condition in Eq. (3) using the PO mechanism, hence a higher budget should be provided. In this case $Q_s^* < Q_s^{\circ}$.

b) FPS local scheduler.: A subsystem S_s is exactly schedulable iff in addition to Eq. (5), $\forall \tau_i, 0 < \forall t \leq D_i, \;\; \mathtt{rbf}_{\mathsf{FP}}(i,t) + b_i \geq \mathtt{sbf}(t)$ (see Theorem 2.3 in [13]). This means that if the budget Q_s is the minimum required budget to guarantee the schedulability of tasks in S_s, then there is a set of times t^f at which $\mathtt{rbf}_{\mathsf{FP}}(i,t) + b_i = \mathtt{sbf}(t)$. Note that Eq. (24) is valid since it is independent on the type of scheduler used.

If all elements in t^f are not in the range given for case 1, then $\forall \tau_i, 0 < \forall t \leq D_i, \;\; \mathtt{rbf}_{\mathsf{FP}}(i,t) + b_i > \mathtt{sbf}(t)$ which makes the local scheduler . To solve this problem, the budget when using the PO mechanism should be increased. In this case $Q_s^* < Q_s^{\circ}$.

■

Lemma 2: Assuming that the minimum required budget to schedule all tasks in a subsystem S_s using the BO mechanism is $Q_s^{\#}$, and that the corresponding budget when using the EO mechanism is Q_s^*, then $Q_s^* = Q_s^{\#}$.

Proof:

Using the EO mechanism or the BO mechanism, the BD time of a subsystem S_s equals to $2P_s - 2Q_s$ see Figure 3. That means the $sbf(t)$ for both mechanisms will be the same for all $t \geq 0$. The demand bound function $\mathtt{dbf}_{\mathsf{EDF}}(i,t)$ when using an EDF scheduler locally will not be changed when using either of the EO mechanism or the BO mechanism. The same goes for the request bound function $\mathtt{rbf}_{\mathsf{FP}}(i,t)$ when using FPS locally. Looking at the Eq. (3) and Eq. (5) then the local schedulability analysis when using the EO mechanism and the BO mechanism will be the same, i.e., $Q_s^* = Q_s^{\#}$.

■

B. System-level comparison

As shown in the previous section, the minimum required budget when using the EO mechanism is equal to the minimum required budget when using the BO mechanism, and lower than or equal to the minimum required budget when using the PO mechanism. However, at system level, it is not easy to see which one of these three approaches that will require minimum overall system CPU resources in the general case.

In doing a comparison among the three approaches, *system load* is defined as a quantitative measure to represent the minimum amount of CPU allocations necessary to guarantee the schedulability of the system S. Then, the impact of each overrun mechanism on the system load can be investigated, respectively.

When using EDF as a global scheduler, the system load is computed as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_t \frac{\mathtt{LBF}(t)}{t} \ . \tag{25}$$

Note that $\alpha = \mathsf{load}_{\mathsf{sys}}$ is the smallest fraction of the CPU that is required to schedule all the subsystems in the system S (satisfying Eq. (6)) assuming that the resource supply function (at system level) is αt.

When using FPS as a global scheduler, the system load is computed as follows:

$$\mathsf{load}_{\mathsf{sys}} = \max_{\forall S_s \in \mathcal{S}} \{\alpha_s\}. \tag{26}$$

where

$$\alpha_s = \min_{0<t\leq P_s} \{\frac{\mathtt{LBF}_s(t)}{t} \mid \mathtt{LBF}_s(t) \leq t\}. \tag{27}$$

Looking at Eq. (25) and Eq. (26), $\mathsf{load}_{\mathsf{sys}}$ can be decreased by lowering $\mathtt{LBF}(t)$.

c) EDF global scheduler. : Comparing between the three overrun mechanisms, the mechanism that requires the lowest $\mathtt{DBF}(t)$ at the time t when $\mathtt{LBF}(t)/t$ is at its maximum will require less system load. Three cases can be distinguished based on the type of overrun mechanism used and its associated demand bound function:

1) **PO vs BO**. Comparing Eq. (14) and Eq. (17), it can be concluded that $\mathtt{DBF}^{\circ}_s(t) \geq \mathtt{DBF}^{\#}_s(t)$ for $0 \leq t < 2 \cdot P_s$. The reason for this is that according to Lemma 2 and Lemma 1, $Q^{\circ}_s \geq Q^{\#}_s$.

When t is in the range of $0 \leq t < 2 \cdot P_s$, the floor in Eq. (14) and Eq. (17) will equal to 0 or 1, which makes Eq. (14) and Eq. (17) identical, and the only difference is the value of the budget. If t is not in this range then it is not possible to decide which mechanism that can give a lower demand bound function without knowing the full interface parameters using both mechanisms. At a certain time instance t^s when $t^s \gg 2P_s$, $\mathtt{DBF}^{\circ}_s(t) < \mathtt{DBF}^{\#}_s(t)$ for $t \geq t^s$.

2) **BO vs EO**. Comparing Eq. (17) and Eq. (19), it can be concluded that $\mathtt{DBF}^{*}_s(t) \geq \mathtt{DBF}^{\#}_s(t)$ for $0 \leq t < 2 \cdot P_s$. The reason for this is that according to Lemma 2 $Q^{*}_s = Q^{\#}_s$, while the floor part in Eq. (17) and Eq. (19) is different. Looking at the EO mechanism, the demand bound function is increased when $t = P_s - H_s$. For the BO mechanism, the demand bound function is increased at $t = P_s$, which means that $\mathtt{DBF}^{*}_s(t) > \mathtt{DBF}^{\#}_s(t)$ at $P_s - H_s \leq t < P_s$. However, if $t > 2P_s$, then it is not possible to decide which one of the two mechanisms that will be better than the other, as in the first case above.
3) **PO vs EO**. Comparing Eq. (14) and Eq. (19), it can be concluded that $\mathtt{DBF}^{\circ}_s(t) < \mathtt{DBF}^{*}_s(t)$ when t is in the range $kP_s - H_s \geq t < kP_s$ and $\mathtt{DBF}^{\circ}_s(t) \geq \mathtt{DBF}^{*}_s(t)$ when t is in $kP_s \geq t < (k+1)P_s - H_s$, where k is an integer value greater and $k > 0$. Note that, at a certain time instance t^s when $t^s \gg 2P_s$, $\mathtt{DBF}^{\circ}_s(t) \geq \mathtt{DBF}^{*}_s(t)$ for $t \geq t^s$ if $Q^{*}_s < Q^{\circ}_s$.

The example shown in Figure 5 explains the three cases described above.

Defining the time t^l as the time at which the system load is evaluated from Eq. (25), then, depending on the value of t^l and the type of overrun mechanism used, it would be possible to estimate which one of the three overrun mechanisms that will require the lowest system load. For example, if $t^l \in 2 \cdot P_k$ and P_k is the shortest subsystem period, then the subsystem load when using the BO mechanism is less than or equal to the subsystem load when using any of the other two overrun mechanisms. However, if $t^l \gg P_k$ then the possibility of having good results when using the BO mechanism is very low. Another aspect that can be considered is when $Q^{\circ}_s = Q^{\#}_s = Q^{*}_s$ for all subsystems, then the system load using the PO mechanism will always be less than or equal to the system load when using the other two mechanisms. Otherwise, all subsystem parameters should be given in order to evaluate which one of the three mechanisms that can give better results in terms of lowest system load.

d) FPS global scheduler. : Looking at Eq. (27), in order to minimize the system load $\mathtt{LBF}_k(t)$ of the subsystem S_k that generates the maximum α should be minimized. The overrun mechanism that generates the lowest request bound function $\mathtt{RBF}_s(t)$ for the subsystem S_k, will require the lowest system load. However, S_k may not be the same subsystem when using different overrun mechanisms,

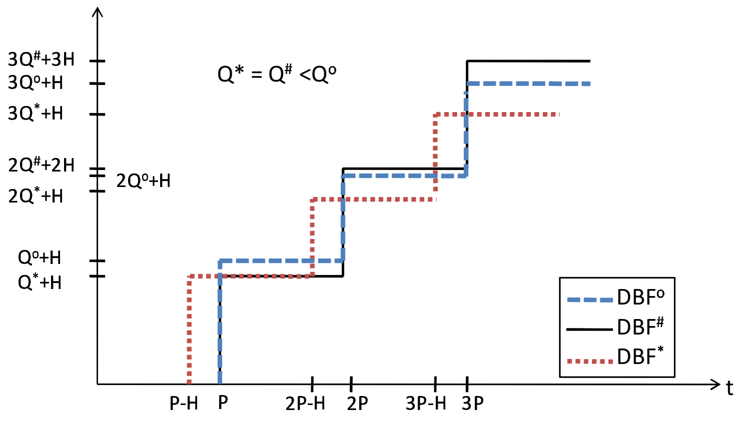

Fig. 5. Comparing between $\mathtt{DBF}_s^{\circ}(t)$, $\mathtt{DBF}_s^{\#}(t)$ and $\mathtt{DBF}_s^{*}(t)$.

and also, at a certain time instance t, the value of $\mathtt{RBF}_s(t)$ when using one of the overrun mechanisms will be less than when using another overrun mechanism, and for another time instance t the value of $\mathtt{RBF}_s(t)$ might be less when using the second mechanism. It can be concluded that none of the three overrun mechanisms can perform better than the other two in the general case, as it depends directly of the system parameters.

The comparison between the three overrun mechanisms in terms of request bound function is shown below;

1) **PO vs BO**. Comparing Eq. (16) and Eq. (18), it easy to show that $\mathtt{RBF}_s^{\circ}(t) \geq \mathtt{RBF}_s^{\#}(t)$ for $0 \leq t < P_s$. The reason is that the interference from other higher priority tasks is always $Q_k + H_k$ for both cases and $Q_s^{\circ} \geq Q_s^{\#}$. If $t > P_s$ then the mechanism that require a lower request bound function is different depending on the system parameters. It can be concluded that if the subsystem periods of all subsystems are equal, then the BO mechanism will require less (or at least equal) system load than using the PO mechanism. Another interesting observation is that if the subsystem that generates maximum α in Eq. (27) has the highest priority, then the BO mechanism will require less (or at least equal) system load than using the PO mechanism. The reason for this is inherent in the subsystem priority; as the subsystem has higher priority, then there will be no interference from other lower priority subsystems.
2) **BO vs EO**. Comparing Eq. (18) and Eq. (21), it is easy to show that $\mathtt{RBF}_s^{*}(t) \geq \mathtt{RBF}_s^{\#}(t)$ for $0 \leq t < P_s$. If $t > P_s$, then finding the best mechanism that requires the least system load

depends on the system parameters.

3) **PO vs EO**. Comparing Eq. (16) and Eq. (21), it can be concluded that $\texttt{RBF}^{\circ}_{s}(t) < \texttt{RBF}^{*}_{s}(t)$ when t is in the range $kP_s - H_s \geq t < kP_s$ and $\texttt{RBF}^{\circ}_{s}(t) \geq \texttt{RBF}^{*}_{s}(t)$ when t is in $(k-1)P_s \geq t < (k)P_s - H_s$ where k is an integer value greater and $k > 0$.

The following examples show some of the cases discussed above:

e) Example 1:: Suppose that a system S consists of three subsystems with parameters as shown below;

Subsystem	Q°	P_s	Q^{*}_{s}=$Q^{\#}$	H
S_1	20	5	4	2
S_2	50	15	13	4
S_3	100	20	18	4

The global scheduler is EDF. Using the PO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.85$ and maximum α is at $t = 100$, using the BO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.86$ and maximum α is at $t = 100$, and for the EO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.755$ and maximum α is at $t = 98$.

f) Example 2:: Suppose that a system S consists of three subsystems with parameters as shown below;

Subsystem	Q°	P_s	Q^{*}_{s}=$Q^{\#}$	H
S_1	12	2	1.75	1
S_2	15	13	2.9	2
S_3	60	10	9.5	3

The global scheduler is EDF. Using the PO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.73$ and maximum α is at $t = 15$, using the BO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.8$ and maximum α is at $t = 60$, and for the EO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.82$ and maximum α is at $t = 13$.

g) Example 3:: Suppose that a system S consists of three subsystems with parameters as shown below;

Subsystem	Q°	P_s	Q_s^{*}=$Q^{\#}$	H	Priority
S_1	40	5	4.5	1	High
S_2	40	2	1.75	1	Middle
S_3	40	3.5	3	2	Low

The global scheduler is FPS. Using the PO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.36$ and maximum α is at $t = 40$, using the BO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.33$ and maximum α is at $t = 40$, and for the EO mechanism $\mathsf{load}_{\mathsf{sys}} = 0.35$ and maximum α is at $t = 38$.

VII. Computing resource holding time

This section explains how to compute the resource holding time $h_{j,i}$, a very important parameter in the global analysis. Using the periodic virtual processor model, each subsystem S_s receives CPU resources with allocation time Q_s every period P_s. During Q_s, the CPU allocation is 100 % of the CPU capacity (see Figure 2 where the slope in the supply curve during Q is one). The mechanism presented in Section V guarantees that locking and releasing a critical section of a globally shared resource R_j will happen within the allocated CPU resource $Q_s+\theta$. Then $h_{j,i}$ will include the execution time of the task τ_i that locks R_j inside the critical section as well as the interference from all tasks within the same subsystem that can preempt the execution inside the critical section. The worst case scenario happens when all tasks that can preempt the execution of the critical section will be released just after task τ_i has entered the critical section of resource R_j.

The resource holding time can be computed depending on the local scheduling algorithm, as shown below;

Under FPS scheduling the resource holding time $h_{j,i}$ of a shared resource R_j is [9];

$$W_j^{FPS}(t) = cx_j + \sum_{\tau_k \in U}^{n} \lceil \frac{t}{T_k} \rceil \cdot C_k, \tag{28}$$

where $cx_{j,i} = \max\{c_{i,j}\}$ is the maximum execution time of task τ_i inside the critical section of the resource R_j and n is the number of tasks and U is the set of tasks such that $U = \{\tau_k | \pi_k > rc_j\}$.

The resource holding time $h_{j,i}$ is the smallest positive time t^{*} such that

$$W_j^{FPS}(t^{*}) = t^{*}. \tag{29}$$

Under EDF scheduling the resource holding time $h_{j,i}$ of a shared resource R_j accessed by task τ_i is [16];

$$W_j^{EDF}(t) = cx_{j,i} + \sum_{\tau_k \in U}^{n} \Big(min\Big(\Big\lceil \frac{t}{T_k} \Big\rceil, \Big\lfloor \frac{D_i - D_k}{T_k} \Big\rfloor + 1 \Big) \Big) \cdot C_k, \tag{30}$$

The resource holding time $h_{j,i}$ is the smallest positive time t^* such that

$$W_j^{EDF}(t^*) = t^*. \tag{31}$$

Finally, the resource holding time of a resource R_j is $h_j = \max\{h_{j,i}\}$ for all τ_i access resource T_j.

Note that the preemption inside the critical section from other subsystems has not been counted when calculating the resource holding time. However, the interference from higher preemption level subsystems is taken into account in the global schedulability analysis. Looking at Eq. (6) and Eq. (9), $h_{j,i}$ may act as a blocking to other subsystems. Moreover, it is also the extra capacity required to prevent budget expiry inside critical section. When $h_{j,i}$ is considered as a blocking time for other subsystems, the effect of interference from higher preemption level subsystems inside the critical section will be included in the global schedulability of the blocked subsystem (in the summation part of Eq. (6) and of Eq. (9)). When $h_{j,i}$ is used to evaluate the overrun, interference from other subsystems inside the critical section will not be important, as the only important part here is that the locked resource should be released before the end of the period.

Eq. (28) and Eq. (30) can be simplified to evaluate $h_{j,i}$ as shown below:

$$h_{j,i} = cx_{j,i} + \sum_{\tau_k \in U} C_k \tag{32}$$

The difference between the simplified equation Eq. (32) and the original equations Eq. (28) and Eq. (30) is that in Eq. (28) and Eq. (30) all tasks that can preempt inside the critical section are assumed to be execute only once. The reason for why it is safe to assume only one execution of each preempting task inside the critical section is given in the following lemma, showing that if a task executes more than one time inside the critical section, the subsystem will become unschedulable.

Lemma 3: For a subsystem that uses an overrun mechanism to arbitrate access to a global shared resource under the periodic virtual processor model, each task that is allowed to preempt the execution of anther task currently inside the critical section of a globally shared resource can, in the worst case, only execute (cause interference) once independent if the local scheduler is EDF or FPS.

Proof: This lemma is proven by considering two cases:

(1) $P_s < T_m$ (where $T_m = min(T_i)$ for all $i = 1, ..n$), if the task having period T_m executes 2 or more times inside the critical section, this means that the resource will be locked during this period, i.e., $h_{j,i} > T_m$ then $h_{j,i} > P_S$, which in turn means that the CPU utilization required by the subsystem S_s will be $U_s = (Q_s + h_{j,i})/P_s > 1$.

(2) If $P_s \geq T_m$, $\mathtt{sbf}_\Gamma(t)$ should provide at least C_m at time $t = T_m$ to ensure the schedulability test in Eq. (3) for the EDF scheduler and in Eq. (4) for the FPS scheduler. Note that $\mathtt{sbf}_\Gamma(t) = 0$ during $t \in [0, 2P_s - 2Q_s]$ so, $2P_s - 2Q_s < T_m$ which means $Q_s > P_s - T_m/2$.

If the task that has period T_m execute 2 times inside the critical section then $h_{j,i} > T_m$. Hence, $Q_s + h_{j,i} > P_s + T_m/2$ which means $U_s = (Q_s + h_{j,i})/p > 1$. ■

From Lemma (3), it can be concluded that if $t^* > T_m$ then the required CPU utilization U_s will be greater than one. This means that, in turn, all tasks that can preempt the execution of a critical section should do so maximum one time in order to keep the utilization of a subsystem less than one. This proves the correctness of Eq. (32) which is based on the assumption that all tasks can interfere only once as a worst case while a task is in the critical section of the resource R_j. If the value of $h_{j,i}$ becomes greater than $min(T_m, P_s)$ then it can be concluded that the subsystem will not be schedulable and no further calculation towards finding a exact value of $h_{j,i}$ is needed.

VIII. Summary

This paper presented three different overrun mechanisms that all can handle the problem of sharing of logical resources in a hierarchical scheduling framework while at the same time supporting independent subsystem development (open environments). Compared to the previous work [8], the results have been generalized by also allowing for the FPS scheduling algorithm for both local and global schedulers, which is suitable to open environment. In addition, a third overrun mechanism, basic overrun without payback (BO), is included in the comparison between the overrun mechanisms. Also, this comparison is performed considering both FPS and EDF scheduling algorithms. The results from comparison showed that it is not trivial to evaluate, in the general case, which overrun method that is better than the other, as their impact on the CPU utilization is highly dependent on global system parameters such as subsystem periods and budgets. Finally, the calculation of resource holding times when using the periodic virtual processor model with both the EDF and the FPS scheduling algorithm is presented, as the resource holding time is a very important parameter in the global schedulability analysis.

Future work includes comparing the enhanced overrun mechanism (EO) with other synchronization mechanisms such as BWI [23], the BROE server [15] and SIRAP [7]. In addition, implementing the three overrun mechanisms and comparing the implementation overhead of each mechanism is important. Finally, as the global schedulability analysis gives an upper bound for EO, it will be interesting to find an exact or less pessimistic schedulability analysis.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the* 19^{th} *IEEE International Real-Time Systems Symposium (RTSS'98)*, pages 4–13, Madrid, Spain, December 1998.

[2] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM international conference on Embedded software (EMSOFT '04)*, pages 95–103, Pisa, Italy, September 2004.

[3] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[4] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the* 11^th *IEEE International Real-Time Systems Symposium(RTSS'90)*, pages 182–190, Lake Buena Vista, Florida, USA, December 1990.

[5] S. K. Baruah. Resource sharing in EDF-scheduled systems: A closer look. In *Proceedings of the* 27^{th} *IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 379–387, Rio de Janeiro, Brazil, December 2006.

[6] M. Behnam, T. Nolte, M. Åsberg, and I. Shin. Synchronization protocols for hierarchical real-time scheduling frameworks. In *Proceedings of the 1st Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08) in conjunction with the 29th IEEE International Real-Time Systems Symposium (RTSS'08)*, November 2008.

[7] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Sirap: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *Proceedings of the 7th ACM and IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288, Salzburg, Austria, October 2007.

[8] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Scheduling of semi-independent real-time components: Overrun methods and resource holding times. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*. IEEE Industrial Electronics Society, September 2008.

[9] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the 15th International Workshop on Parallel and Distributed Real-Time Systems(WPDRTS)*, pages 1–8, Long Beach, CA, USA, March 2007.

[10] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the* 26^{th} *IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 389–398, Miami Beach, FL, USA, December 2005.

[11] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the*

27^{th} *IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 389–398, Rio de Janeiro, Brazil, December 2006.

[12] Z. Deng and J.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the* 18^{th} *IEEE International Real-Time Systems Symposium (RTSS'97)*, pages 308–319, San Francisco, CA, USA, December 1997.

[13] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the* 28^{th} *IEEE International Real-Time Systems Symposium(RTSS'07)*, pages 129–138, 2007.

[14] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* 23^{th} *IEEE International Real-Time Systems Symposium (RTSS'02)*, pages 26–35, Austin, TX, USA, December 2002.

[15] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In *Proceedings of the* 28^{th} *IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 83–92, December 2007.

[16] N. Fisher, M. Bertogna, and S. Baruah. Resource-locking durations in EDF-scheduled systems. In *Proceedings of the* 13^th *IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 91–100, Bellevue, WA, USA, 2007.

[17] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of the* 20^{th} *IEEE International Real-Time Systems Symposium (RTSS'99)*, pages 256–267, Phoenix, AZ, USA, December 1999.

[18] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the* 20^{th} *IEEE International Real-Time Systems Symposium(RTSS'89)*, pages 166–171, Santa Monica, CA, USA, December 1989.

[19] G. Lipari and S. K. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proceedings of the* 6^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 166–175, May-June 2000.

[20] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of the* 15^{th} *Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 151–158, Porto, Portugal, July 2003.

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

[22] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of the* 21^{th} *IEEE International Real-Time Systems Symposium(RTSS'00)*, pages 217–226, Orlando, FL, USA, December 2000.

[23] G. Lipari, G. Lamastra, and L. Abeni. Task synchronization' in reservation-based real-time systems. *IEEE Transactions on Computers*, 53(12):1591–1601, December 2004.

[24] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium(RTAS)*, pages 75–84, Taipei, Taiwan ROC, May 2001.

[25] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the* 9^{th} *IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, Huntsville, AL, USA, December 1988.

[26] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarhical fixed-priority scheduling. In *Proceedings of the* 14^{th} *Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 152–160, Vienna, Austria, June 2002.

[27] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the International Conference on Industrial Electronics, Control, and Instrumentation IECON87*, pages 909–916, Cambridge, MA, USA, November 1987.

[28] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the* 24^{th} *IEEE International Real-Time Systems Symposium(RTSS'03)*, pages 2–13, Cancun, Mexico, December 2003.

[29] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of the* 25^{th} *IEEE International Real-Time Systems Symposium(RTSS'04)*, pages 57–67, Lisbon, Portugal, December 2004.

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