An Overrun Method to Support Composition of Semi-Independent Real-Time Components

Moris Behnam, Insik Shin, Thomas Nolte, Mikael Nolin
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
email: moris.behnam@mdh.se

Abstract

Engineers of embedded software systems rely on efficient design techniques and tools along with efficient run-time support. In the design of complex embedded real-time systems, the Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework enabling compositional schedulability analysis of embedded software systems with real-time properties. Moreover, the HSF provides a run-time framework guaranteeing that these non-functional requirements are met.

In this paper a system consists of a number of semi-independent components called subsystems, and these subsystems are allowed to share logical resources. The HSF makes sure that the individual subsystems respect their allocated CPU budgets. However, as semi-independent subsystems share logical resources, extra complexity is introduced. Specifically, the contribution of this paper is a novel method to allow for budget overruns; a common scenario when a subsystem utilizes shared logical resources. This proposed method is not only more resource efficient than existing methods, but it is also more appropriate for supporting composability of independently developed real-time subsystems.

1 Introduction

Component based design has been accepted as a novel methodology for designing and developing complex embedded software systems through systematic abstraction and composition. The Hierarchical Scheduling Framework (HSF) provides means for decomposing a complex system into well-defined parts [14]. 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. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal behavior. Also, the HSF facilitates reusability of subsystems in timing-critical and resource constrained environments, since the well defined interfaces characterize their computational requirements.

The HSF can be used to support multiple applications while guaranteeing independent execution of those applications. This can be correctly achieved when the system provides partitioning, where the applications may be separated functionally for fault containment and for compositional verification, validation and certification. The HSF provides such a partitioning by making sure that subsystems do respect their allocated budgets, preventing one partitioned function from causing a failure of another partitioned function in the time domain.

Earlier efforts have been made in supporting compositional subsystem integration in hierarchical scheduling frameworks, preserving the independently analyzed schedulability of individual subsystems. One of the more common assumptions 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 logical resource sharing.

To enable logical resource sharing in the HSF, a recent study [6] proposed the overrun mechanism that allows the subsystem to overrun (its budget) to complete the execution of the critical section. The study provided schedulability analysis for this mechanism; however, it does not allow independent analysis of individual subsystems. Hence, this schedulability analysis does not support composability. For this overrun mechanism, we present schedulability analysis supporting composability. Furthermore, we propose an enhanced overrun mechanism that gives two benefits.
(compared with the existing mechanisms): (1) it increases schedulability within a subsystem by providing CPU allocations more efficiently, and (2) it can even accept subsystems which developed their timing requirements without knowing that the proposed overrun mechanism would be employed in the system.

The outline of the paper is as follows: Section 2 presents related work, and Section 3 provides some backgrounds and system model. Section 4 presents schedulability analysis. Section 5 introduces the overrun mechanisms, and finally, Section 6 concludes.

2 Related work

This section presents related work in the areas of hierarchical scheduling frameworks as well as resource sharing protocols.

2.1 Hierarchical scheduling

For real-time systems, there has been a growing attention to hierarchical scheduling frameworks, in particular, to the hierarchical schedulability analysis with the periodic resource model. This periodic resource model can specify the periodic resource allocation guarantees provided to a component from its parent component [14]. Saewong et al. [12] and Lipari and Bini [9] introduced schedulability conditions for fixed-priority local scheduling, and Shin and Lee [14] presented a schedulability condition for EDF local scheduling. Davis and Burns [5] evaluated different periodic servers (Polling, Deferrable, and Sporadic Servers) for fixed-priority local scheduling.

2.2 Resource sharing

When several tasks are sharing a logical resource, typically only one task is allowed to use the resource at a time. Thus the logical resource requires mutual exclusion of tasks that uses it. To achieve this a mutual exclusion protocol is used. The protocol provides rules about how to gain access to the resource, and specifies which tasks should be blocked when trying to access the resource. To achieve predictable real-time behaviour, several protocols have been proposed including the Priority Inheritance Protocol (PIP) [13], the Priority Ceiling Protocol (PCP) [11], and the Stack Resource Policy (SRP) [2]. Furthermore, there have been studies on supporting resource sharing in hierarchical scheduling frameworks [7, 8, 1, 10]. Recently, Davis and Burns [6] proposed the Hierarchical Stack Resource Policy (HSRP) to support logical resource sharing. However, their work does not support composability, disallowing independent analysis of individual subsystems.

3 System model and background

3.1 Resource sharing under hierarchical scheduling

A hierarchical scheduling framework is introduced to support CPU time sharing among applications (subsystems) under different scheduling services. The system-level global scheduler allocates CPU time to subsystems, and the subsystem-level local schedulers subsequently schedule CPU time to their internal tasks. This framework also allows logical resource sharing between tasks in a mutually exclusive manner (see Figure 1). Tasks can share local logical resources within a subsystem and global logical resources across subsystems. In this paper we focus on global logical resources while local logical resources can be easily supported by traditional synchronization protocols such as SRP [1, 6, 8].

3.2 Virtual processor models

Shin and Lee [14] proposed the periodic virtual processor model $\Gamma(P, Q)$ 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 $U_\Gamma$ of a periodic virtual processor model $\Gamma(P, Q)$ is defined as $Q/P$.

The supply bound function $sbf_\Gamma(t)$ of the periodic model $\Gamma(P, Q)$ was given in [14] to compute the minimum resource supply during an interval of length $t$. In this paper, we rephrase it with an additional parameter of BD, where BD represents its longest possible blackout duration during which the model may provide no resource allocation.

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

where $k = \max \left( \left\lceil \frac{(t + (P - Q) - BD)}{P} \right\rceil, 1 \right)$ and $W^{(k)}$ denotes an interval $[(k - 1)P + BD, (k - 1)P + BD + Q]$. 

![Figure 1. Two-level hierarchical scheduling framework with resource sharing.](image-url)
Here, we first note that the original $sbf(t)$ in [14] is equivalent to $sbf_1(t, BD)$ when BD = 2($P - Q$). We 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 model $\Gamma(P,Q)$. Figure 2 illustrates the supply bound function $sbf_1(t)$.

3.3 Stack resource policy (SRP)

To use SRP [2] in a hierarchical framework, we extend terms associated with SRP as follows:

- **Preemption level.** Each task $\tau_i$ has a preemption level equal to $\pi_i = 1/D_i$, where $D_i$ is a relative deadline. Similarly, each subsystem $S_s$ has a preemption level equal to $\Pi_s = 1/P_s$, where $P_s$ is the subsystem’s period.

- **Resource ceiling.** Each global shared resource $R_j$ is associated with two types of resource ceilings; an internal resource ceiling for local scheduling $rc_j = \max\{\pi_i|\tau_i \text{ accesses } R_j\}$ and an external resource ceiling for global scheduling.

- **System and subsystem ceilings.** System and subsystem ceilings are dynamic parameters that change during execution. The system (subsystem) ceiling is equal to the currently locked highest external (internal) resource ceiling in the system (subsystem).

According to SRP, a job $J_i$ generated by task $\tau_i$ can preempt the currently executing job $J_k$ within a subsystem only if $J_i$ is a higher-priority job of $J_k$ and the preemption level of $\tau_i$ is greater than the current subsystem ceiling. A similar reasoning is made for subsystems from a global scheduling point of view.

3.4 System model

We consider a periodic task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$, 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 $\tau_i$. Each element $c_{i,j}$ in $\{c_{i,j}\}$ represents the WCET of the task $\tau_i$ inside a critical section of the global shared resource $R_j$.

For a shared resource $R_j$, the resource holding time $h_{j,i}$ of a task $\tau_i$ is defined as the maximum task execution time inside a critical section plus the interference (inside the critical section) of higher priority tasks that have preemption level greater than the internal ceiling of the locked resource.

A subsystem $S_s \in S$, where $S$ is the whole system of subsystems, is characterized by a task set $T_s$ and a set of internal resource ceilings $RC_s$ of the global shared resources. Each subsystem $S_s$ is assumed to have an EDF local scheduler. Each subsystem $S_s$ has an interface (the subsystem interface) that is defined as $(P_s, Q_s, H_s)$, where $P_s$ is a period, $Q_s$ is an execution requirement budget, and $H_s$ is a maximum global resource holding time, i.e., $H_s = \max\{h_{j,i}|\tau_i \in T_s \text{ accesses } R_j\}$.

4 Schedulability analysis

This section presents the schedulability analysis when using HSF. Starting with local schedulability analysis, we then present how to calculate subsystem interfaces and finally, global schedulability analysis is presented. The analysis presented assumes that SRP are used for synchronization on the local level.

4.1 Local schedulability analysis

Let $dbf_{EDF}(i, t)$ denote the demand bound function of a task $\tau_i$ under EDF scheduling [3], i.e.,

$$dbf_{EDF}(i, t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \cdot C_i.$$  \hspace{1cm} (2)

The local schedulability condition under EDF scheduling is then (by combining the results of [4, 14])

$$\forall t > 0 \sum_{i=1}^{n} dbf_{EDF}(i, t) + b(t) \leq sbf(t),$$  \hspace{1cm} (3)

where $b(t)$ is the blocking function [4] 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.
4.2 Subsystem interface calculation

In this section, we explain how to derive the budget \( Q_s \) of the subsystem interface. Given \( S_s, RC_s, \) and \( P_s, \) let \( \text{calculateBudget}(S_s, P_s, RC_s) \) denote a function that calculates the smallest subsystem budget that satisfies Eq. (3). Such a function is similar to the one in [14]. Then, \( Q_s = \text{calculateBudget}(S_s, P_s, RC_s). \)

4.3 Global schedulability analysis

Global schedulability analysis under EDF scheduling is given with the system load bound function \( \text{LBF}(t) \) as follows (by Theorem 1 of [4]):

\[
\forall t > 0 \quad \text{LBF}(t) = B(t) + \sum_{S_s \in S} \text{DBF}_s(t) \leq t, \tag{4}
\]

where

\[
\text{DBF}_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor Q_s, \tag{5}
\]

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 \( P_k > t, \) and is defined as

\[
B(t) = \max\{H_k \mid P_k > t\}. \tag{6}
\]

5 Overrun mechanisms

This section explains overrun mechanisms that can be used to handle budget expiry during a critical section in a hierarchical scheduling framework. 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 tasks within the same subsystem can initiate 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 can cause a problem, if it happens while a job \( J_i \) of a subsystem \( S_s \) is executing within the 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 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, we consider a mechanism based on overrun that works as follows; when the budget of subsystem \( S_s \) expires and \( S_s \) has a job \( J_i \) that is still locking a global 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 \textit{overrun time} \( \theta. \) The maximum \( \theta \) occurs when \( J_i \) lock a resource that gives the longest resource holding time just before the budget of \( S_s \) expires. Here, we consider the payback overrun mechanism [6]. Whenever overrun happens, the subsystem \( S_s \) pays back \( \theta \) in the next execution instant, i.e., the subsystem budget \( Q_s \) will be decreased by \( \theta \) for the subsystem execution instant that follows the overrun (note that only the instant following the overrun is affected). Hereinafter, we call this payback overrun mechanism \textit{basic overrun}.

5.1 Basic overrun

Davis et al. [6] presented schedulability analysis with the basic overrun, however, it is not suitable for open environments [7] 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 suitable with the basic overrun mechanism in open environments.
5.1.1 Independent schedulability analysis with basic overrun

The supply bound function in [14] was developed under the assumption that the greatest blackout duration is $2(P - Q)$. The basic overrun cannot employ this existing supply bound function for schedulability analysis because the greatest Blackout Duration (BD) is $2(P - Q) + H$ with the basic overrun (as shown in Figure 3(a)). Taking this into account, this paper presents a modified supply bound function $\text{sbf}_t^*(t)$, that can be used with the basic overrun (using Eq. (1)), as follows:

$$\text{sbf}_t^*(t) = \text{sbf}_t(t, \text{BD}^\circ), \quad \text{where } \text{BD}^\circ = 2(P - Q) + H. \tag{7}$$

We can then extend the existing schedulability conditions of Eq. (3) by substituting $\text{sbf}_t(t)$ with $\text{sbf}_t^*(t)$.

5.1.2 Global schedulability analysis with basic overrun

We first discuss how to extend the demand bound function of a subsystem with the basic overrun mechanism. Looking at the basic overrun with payback in a subsystem $S_s$, the maximum contribution on $\text{DBF}_t^*(t)$ 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 $Q_s - H_s + H_s = Q_s$. Hence, the demand bound function $\text{DBF}_t^*(t)$ of a subsystem $S_s$ with the basic overrun mechanism is

$$\text{DBF}_t^*(t) = \text{DBF}_t(t) + O_s(t), \tag{8}$$

where

$$O_s(t) = \begin{cases} H_s & \text{if } t \geq P_s, \\ 0 & \text{otherwise}. \end{cases} \tag{9}$$

We can then extend the schedulability condition of Eq. (4) by substituting $\text{DBF}_t(t)$ with $\text{DBF}_t^*(t)$.

5.2 Enhanced overrun

As seen in Section 5.1, the basic overrun mechanism works with the modified supply bound function $\text{sbf}_t^*(t)$ that is less efficient in CPU resource usage compared with the original $\text{sbf}_t(t)$, as illustrated in Figure 4. We propose an enhanced overrun mechanism that makes it possible to use $\text{sbf}_t(t)$ with overrun to improve the efficiency of CPU resource utilization.

The enhanced overrun mechanism is based on imposing an offset (delaying the budget replenishment of subsystem) equal to the amount of an overrun $\theta_s$ to the execution instant that follows a subsystem overrun. As shown in Figure 3(b), the maximum BD is $2(P - Q)$ and therefore it is possible to use the existing supply bound function [14]. As a result of minimizing the maximum BD, the subsystem budget $Q_s^\circ$ required to guarantee the schedulability of the subsystem tasks will be less compared with the subsystem budget if we use basic overrun mechanism. Another important features that the enhanced overrun 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 an overrun mechanism or not. This feature is very important in an open environment. We can then use the existing local EDF schedulability condition of Eq. (3) without any modification.

5.2.1 Global schedulability analysis with enhanced overrun

The effect of overrun is now moved to global scheduling analysis where the demand bound function $\text{DBF}_t^*(t)$ of a subsystem $S_s$ is changed to include the offset $\theta_s = H_s$ as shown

$$\text{DBF}_t^*(t) = \left[ \frac{t + H_s}{P_s} \right] Q_s^\circ + O_s^\circ(t), \tag{10}$$

where

$$O_s^\circ(t) = \begin{cases} H_s & \text{if } t \geq P_s - H_s, \\ 0 & \text{otherwise}. \end{cases} \tag{11}$$

6 Summary

In this paper we have proposed a new overrun mechanism, for hierarchical scheduling frameworks, that can be used in the domain of open environments. We have presented both independent local schedulability analysis as well as global schedulability analysis for the proposed overrun mechanism as well as the existing basic overrun.

Future work includes the development of local and global schedulability analysis towards fixed-priority scheduling. We also intend to compare the previous basic overrun and the proposed enhance overrun, in terms of the minimum resource requirements to guarantee global schedulability.

References


