Overrun and Skipping in Hierarchically Scheduled Real-Time Systems

Moris Behnam, Thomas Nolte, Mikael Åsberg
Mälardalen Real-Time Research Centre
P.O. Box 883, SE-721 23 Västerås, Sweden
moris.behnam@mdh.se

Reinder J. Bril
Technische Universiteit Eindhoven (TU/e)
Den Dolech 2, 5612 AZ Eindhoven
The Netherlands

Abstract

Recently, two SRP-based synchronization protocols for hierarchically scheduled real-time systems based on Fixed-Priority Preemptive Scheduling (FPPS) have been presented, i.e., HSRP [9] and SIRAP [4]. Preventing depletion of budget during global resource access, the former implements an overrun mechanism, while the later exploits a skipping mechanism. A theoretical comparison of the performance of these mechanisms revealed that none of them was superior to the other, as their performance is heavily dependent on the system's parameters. To better understand the relative strengths and weaknesses of these mechanisms, this paper presents a comparative evaluation of the depletion prevention mechanisms overrun (with or without payback) and skipping. These mechanisms are investigated in detail and the corresponding system load imposed by these mechanisms is explored in a simulation study. The mechanisms are evaluated assuming FPPS and a periodic resource model [23]. The periodic resource model is selected as it supports locality of schedulability analysis, allowing for a truthful comparison of the mechanisms. Given system characteristics, guiding the design of hierarchically scheduled real-time systems, the results of this paper indicate when one mechanism is better than the other and how a system should be configured in order to operate efficiently.

1 Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced as a scheduling approach to support hierarchical CPU sharing among applications under different scheduling services [23]. The HSF 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., tasks), and resources are allocated from a parent node to its children nodes.

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

The HSF provides means for decomposing a complex system into well-defined parts called subsystems. In essence, the HSF provides a mechanism for timing-predictable composition of coarse-grained subsystems. In the HSF a subsystem provides an introspective interface that specifies the timing properties of the subsystem precisely [23]. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal interference. Temporal isolation between subsystems is provided through budgets which are allocated to subsystems.

Motivation: Research on HSFs started with the assumption that subsystems are independent, i.e., inter-subsystem resource sharing other than the CPU fell outside their scope. In some cases [1, 14], intra-subsystem resource sharing is addressed using existing synchronization protocols for resource sharing between tasks, e.g., the Stack Resource Policy (SRP) [2]. Recently, two SRP-based synchronization protocols for inter-subsystem resource sharing in FPPS systems have been presented, i.e., HSRP [9] and SIRAP [4]. An initial comparative assessment of HSF synchronization protocols, based on five criteria, revealed that none of them was superior to the others [3], however. In particular, the performance of the protocol turned out to be heavily dependent on the system parameters.

One of the main differences between these two synchronization protocols is the way they deal with inter-subsystem resource sharing and depletion of budgets. HSRP is based on an overrun mechanism (with or without payback), i.e., upon depletion of the budget during global resource access, the budget is temporally increased with a statically determined amount for the duration of that access, whereas SIRAP is based on a skipping mechanism, preventing depletion of the budget during global resource access. To better understand the relative strengths and weaknesses of synchronization protocols for HSFs, a comparative evaluation of these underlying mechanisms is presented in this paper.

Although HSRP and SIRAP assume a two-level HSF, it is hard to truthfully compare these protocols in their original settings, because they assume different virtual processor models and different scheduling mechanisms.
Therefore, the comparison in this paper is performed using a common virtual processor model, the periodic resource model from [23], and a common local and global scheduling mechanism; Fixed-Priority Preemptive Scheduling (FPPS). Schedulability analysis for the overrun mechanism (of HSRP) and the skipping mechanism (of SIRAP) assuming the periodic resource model and FPPS can be found in [22] and [4], respectively.

**Contributions:** In this paper, the efficiency of the mechanisms is shown by exploring the system load [3] in a simulation study. The theoretical study of system load shows under which system configuration one mechanism is better than the other.

**Outline:** Section 2 presents related work, and Section 3 presents the system model and background. Section 4 presents a simulation study comparing the system load imposed by overrun and skipping. Finally, Section 5 concludes the paper.

## 2 Related work

This section presents related work in the areas of hierarchical scheduling and synchronization protocols.

**Hierarchical scheduling** Over the years, there has been a growing attention to hierarchical scheduling of real-time systems [1, 8, 10, 11, 14, 15, 16, 19, 21, 23]. Deng and Liu [10] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [14] presented schedulability analysis techniques for such a two-level framework with the Fixed-Priority Preemptive Scheduling (FPPS) global scheduler. Lipari and Baruah [15, 18] presented schedulability analysis techniques for Earliest Deadline First (EDF) global schedulers. Mok et al. [20, 11] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [23] introduced the periodic virtual processor model (to characterize the periodic CPU allocation behavior), and many studies have been proposed on schedulability analysis with this model under FPPS [1, 8, 16] and under EDF scheduling [23, 25]. However, a common assumption shared by all above studies is that tasks are independent.

**Synchronization** In order to allow for dependencies among tasks, many synchronization protocols have been introduced for arbitrating accesses to shared logical resources, addressing the priority inversion problem, e.g., the Stack Resource Policy (SRP) [2]. For usage in a HSF, additional protocols have been proposed, e.g., the Hierarchical Stack Resource Policy (HSRP) [9], the Subsystem Integration and Resource Allocation Policy (SIRAP) [4] and the Bounded-delay Resource Open Environment (BROE) [12] protocols.

The work in this paper concerns the former two, targeting systems implementing FPPS schedulers. To bound the waiting time of tasks from different subsystems that want to access the same shared resource, subsystem budget expiration should be prevented while locking a global shared resource. The following two mechanisms can be used to solve this problem:

1. **the overrun mechanism** The problem of subsystem budget depletion inside a critical section is handled by adding extra resources to the budget of each subsystem to prevent the budget expiration inside a critical section. HSRP is based on an overrun mechanism. HSRP stops task preemption within the subsystem whenever a task is accessing a global shared resource. SRP is used at the global level to synchronize the execution of subsystems that have tasks accessing global shared resources. Two versions of overrun mechanisms have been presented: 1) with payback; whenever overrun happens in a subsystem $S_i$, the budget of the subsystem will, in its next execution instant, be decreased by the amount of the overrun time. 2) without payback; no further actions will be taken after the event of an overrun.

2. **the skipping mechanism** Skipping is another mechanism that prevents a task from locking a shared resource by skipping its execution if its subsystem does not have enough remaining budget at the time when the task tries to lock the resource. SIRAP is based on the skipping mechanism. SIRAP uses the SRP protocol to synchronize the access to global shared resources in both local and global scheduling. SIRAP checks the remaining budget before granting the access to the globally shared resources; if there is sufficient remaining budget then the task enters the critical section, and if there is insufficient remaining budget, the local scheduler delays the critical section entering of the job until the next subsystem budget replenishment (assuming that the subsystem budget in the next subsystem budget replenishment is enough to access the global shared resource by the task).

## 3 System model and background

This paper focuses on scheduling of a single node, where each node is modeled as a system $S$ consisting of one or more subsystems $S_i \in S$. The system is scheduled by a two-level Hierarchical Scheduling Framework (HSF) as shown in Figure 1. During runtime, the system level scheduler (global scheduler) selects, at all times, which subsystem will access the common (shared) CPU resource. The synchronization protocols, SRP and HSRP+SIRAP, will mediate access to local and global shared logical resources, respectively.

---

1 The idea of skipping has been firstly considered in the zone based protocol ZB [13] used in a pfair-scheduling environment, while we use it for hard real-time tasks under hierarchical scheduling.
Subsystem model A subsystem $S_s$ consists of a task set $T_s$ of $n_s$ tasks and a scheduler. Once a subsystem is assigned the processor (CPU), its local scheduler will select which subsystem-internal task will be executed. Each subsystem $S_s$ is associated with a subsystem timing interface $S_s(P_s, Q_s, \{X_{s,j}\})$, where $Q_s$ is the subsystem budget that the subsystem $S_s$ will receive every subsystem period $P_s$, and $X_{s,j}$ is the maximum time that a subsystem internal task may lock a shared resource $R_j$. The subsystem interface is used to specify the collective temporal requirements of a subsystem, and it is used as an interface between the subsystem and the global scheduler. Finally, both the local scheduler of a subsystem $S_s$ as well as the global scheduler of the system $S$ are assumed to implement the FPPS scheduling policy. Let $HP(s)$ return the set of subsystems with priorities higher than that of $S_s$ and let $R_s$ be a set of global shared resources accessed by $S_s$. In this paper and for simplicity, $X_s$ is used instead of $\{X_{s,j}\}$ in the subsystem interface. $X_s$ equals to the maximum element in $\{X_{s,j}\}$. Hence, the impact of this simplification will make the results more pessimistic.

Task model The task model considered in this paper is the deadline-constrained sporadic hard real-time task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$, where $T_i$ is a minimum separation time between arrival of successive jobs of $\tau_i$, $C_i$ is their worst-case execution-time, and $D_i$ is an arrival-relative deadline ($0 < C_i \leq D_i \leq T_i$) before which the execution of a job must be completed. Each task is allowed to access one or more shared logical resources, and $c_{i,j}$ is a critical section execution time that represents a worst-case execution-time requirement of task $\tau_i$ within a critical section of a global shared resource $R_j$ (for simplicity of presentation, we assume that each task accesses a shared resource at most one time). It is assumed that all tasks belonging to the same subsystem are assigned unique static priorities and are sorted according to their priorities in the order of increasing priority. Without loss of generality, it is assumed that the priority of a task is equal to the task ID number after sorting, and the greater a task ID number is, the higher its priority is. The set of shared resources accessed by $\tau_i$ is denoted $\{R_i\}$. Let $hp(i)$ return the set of tasks with priorities higher than the priority of $\tau_i$ and $lp(i)$ return the set of tasks with priorities lower than the priority of task $\tau_i$. For each subsystem, we assume that the subsystem period is selected such that $2P_s \leq T_m$, where $T_m$ is the task with the shortest period. This restriction is made for reasons of resource efficiency [24]. Moreover, this assumption simplifies the presentation of the paper (evaluating $X_s$).

Periodic resource model The CPU supply refers to the amount of CPU allocation that a virtual processor can provide. Shin and Lee [23] proposed the periodic processor resource model $\Gamma(P, Q)$ to specify periodic CPU allocations, where $P$ is a period ($P > 0$) and $Q$ is a periodic allocation time ($0 < Q \leq P$).

The supply bound function $sbf_\Gamma(t)$ of $\Gamma(P, Q)$ was given in [23] that computes the minimum possible CPU supply for every interval length $t$ as follows:

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

where $k = \max \left(\left\lfloor \frac{t - (P - Q)}{P} \right\rfloor, 1 \right)$ and $V^{(k)}$ denotes an interval $[(k + 1)P - 2Q, (k + 1)P - Q]$.

3.1 Shared resources

The presented HSF allows for sharing of logical resources between arbitrary tasks, located in arbitrary subsystems, in a mutually exclusive manner. To access a resource $R_j$, a task must first lock the resource, and when the task no longer needs the resource it is unlocked. The time during which a task holds a lock is called a critical section. At any time, only a single task may hold its lock. A resource that is used by tasks in more than one subsystem is denoted a global shared resource. A resource only used within a single subsystem is denoted a local shared resource. The work in this paper targets managing global shared resources, and throughout the remainder of the paper these are simply denoted as shared resources. Management of local shared logical resources can be done by using one of several existing synchronization protocols. In this paper, local shared resources are managed by SRP.
To be able to use SRP in a HSF for synchronizing global shared resources, its associated terms resource, system and subsystem ceilings are extended as follows:

**Resource ceiling** Each global shared resource $R_j$ is associated with two types of resource ceilings; an internal resource ceiling ($rc_j$) for local scheduling and an external resource ceiling ($RX_j$) for global scheduling. They are defined as $rc_j = \max\{i | \tau_i \in T_s \text{ accesses } R_j\}$ and $RX_j = \max\{s | S_s \text{ accesses } R_j\}$. However, assigning an internal resource ceiling according to SRP makes the value of $X_s$ very high which makes the subsystem require more CPU resources. Note that HSRP prevents any task preemption locally (within the subsystem) while accessing shared resources, which can be implemented using SRP with an internal resource ceiling equal to the maximum task priority $rc_j = n_s$. [7] showed that preventing preemption while accessing a global shared resource may violate the local schedulability of the subsystem, and proposed an algorithm based on increasing the ceiling of all resources\(^2\) in steps as much as possible without violating the local schedulability.

**System/subsystem ceiling** The system/subsystem ceilings are dynamic parameters that change during execution. The system/subsystem ceiling is equal to the highest external/internal resource ceiling of a currently locked resource in the system/subsystem.

Under SRP, a task $\tau_k$ can preempt the currently executing task $\tau_i$ (even inside a critical section) within the same subsystem, only if the priority of $\tau_k$ is greater than its corresponding subsystem ceiling. The same reasoning can be made for subsystems from a global scheduling point of view.

### 3.2 Schedulability analysis

In order to be able to compare the overrun and skipping mechanisms when used by synchronization protocols in a HSF, HSRP and SIRAP have been implemented in the same HSF allowing for a fair comparison of their performance. HSRP comes in two flavors, with and without payback, denoted as Overrun With Payback (OWP) and Overrun with No Payback (ONP), respectively. SIRAP implements the Skipping (SKP) mechanism. Central to the discussions later in the paper, the general local and a global schedulability analysis, independent on a particular mechanism, is first presented followed by its modifications for each one of the three mechanisms.

**General local schedulability analysis** The general local schedulability analysis under FPPS is as follows [2, 23]:

\[
\forall \tau_i \exists t : 0 < t \leq D_i, \quad \text{rbf}_i(t) \leq \text{sbf}(t), \quad (2)
\]

where $\text{rbf}_i(t)$ denotes the request bound function of a task $\tau_i$ (later it is shown how to calculate $\text{rbf}_i(t)$, which is dependent on the synchronization protocol in use). Note that $t$ can be selected within a finite set of scheduling points [17].

**General global schedulability analysis** The general condition for global schedulability is

\[
\forall S_s \exists t : 0 < t \leq P_s, \quad \text{RBF}_s(t) + B_s \leq t, \quad (3)
\]

where $\text{RBF}_s(t)$ denotes the request bound function of a subsystem $S_s$ and $B_s$ is the maximum blocking imposed to a subsystem $S_s$, when it is blocked by lower-priority subsystems (suppose that $S_j$ imposes the maximum blocking on $S_s$, then $B_s = X_j$). Note that the way of calculating $\text{RBF}_s(t)$ depends on the synchronization protocol.

**Overrun mechanism without payback** $\text{rbf}_i(t)$ using ONP is calculated as follows [22]:

\[
\text{rbf}_i(t) = C_i + b_i + \sum_{\tau_k \in \text{hp}(i)} \left[ \frac{t}{T_k} \right] \cdot C_k, \quad (4)
\]

where $b_i$ is the maximum blocking imposed to a task $\tau_i$ by lower priority tasks that access resources with ceiling greater than or equal to the priority of $\tau_i$ (i.e., $b_i = \max_{\tau_l \in \text{hp}(i), \tau_l \leq \tau_i} (rc_{k,j})$).

$\text{RBF}_s(t)$ can be calculated as follows:

\[
\text{RBF}_s(t) = (Q_s + X_s) + \sum_{S_k \in \text{BP}(s)} \left[ \frac{t}{P_k} \right] (Q_k + X_k) \quad (5)
\]

**Overrun mechanism with payback** Eq. (4) and Eq. (2) can be used for local schedulability analysis for the OWP mechanism, however, calculating the supply bound function $\text{sbf}(t)$ will be different. Let $\text{sbf}^+(t)$ be the supply bound function when using the overrun mechanism with payback, then $\text{sbf}^+(t)$ can be evaluated as follows [6]:

\[
\text{sbf}^+(t) = \max(\text{sbf}(t) - X_s, 0). \quad (6)
\]

Eq. (7) is used to calculate $\text{RBF}_s(t)$.

\[
\text{RBF}_s(t) = (Q_s + X_s) + \sum_{S_k \in \text{BP}(s)} \left( \left[ \frac{t}{P_k} \right] \cdot Q_k + X_k \right) \quad (7)
\]

\(^2\)Lowering the value of $X_s$ may not always decrease the minimum required CPU resources, it may increase it as shown in [22], however this issue is out of the scope of this paper and left for future work.
calculates the smallest subsystem budget and as little CPU resources as possible. Given a subsystem budget should be computed so that the system will require subsystem period is given while the minimum subsystem

In this paper, it is assumed that the Subsystem budget then not schedulable globally. Then a global shared resource and that means the system is

3.2.1 Calculating \( X_s \)

Given a subsystem \( S_s \), its critical section execution time \( X_s \) represents a worst-case CPU demand that internal tasks of \( S_s \) may collectively request while executing inside any critical section. Note that any task \( \tau_i \) accessing a resource \( R_j \) can be preempted by tasks with priority higher than the internal resource ceiling \( r_{c_j} \) of \( R_j \). Let denote the maximum time that an internal task \( \tau_i \) locks \( R_j \) within the subsystem by \( X_{i,j} \). Note that both HSRP and SIRAP prevent subsystem budget expiration inside a critical section of a global shared resource. To ensure the global schedulability analysis, \( X_{i,j} < P_s \) and since we assume that \( 2P_s \leq T_m \), then all tasks that are allowed to preempt while \( \tau_i \) accesses \( R_j \) will be activated at most one time. Otherwise, only if \( X_{i,j} > P_s \), tasks will be able to execute more than one time while locking a global shared resource and that means the system is not schedulable globally. Then \( X_{i,j} \) can be computed as

\[
X_{i,j} = c_{i,j} + \sum_{k=r_{c_j}+1}^{r_{s}} C_k. \tag{13}
\]

Let \( X_j = \max(X_{i,j}) \) for all \( \tau_i \) accessing resource \( R_j \), then \( X_s = \max(X_j) \) for all \( R_j \in R_s \).

Subsystem budget In this paper, it is assumed that the subsystem period is given while the minimum subsystem budget should be computed so that the system will require as little CPU resources as possible. Given a subsystem \( S_s \), and \( P_s \), let \( \text{calculateBudget}(S_s, P_s) \) denote a function that calculates the smallest subsystem budget \( Q_s \) that satisfies

\[
\text{RBF}_s(t) = Q_s + \sum_{S_k \in \text{RBF}(S)} \left[ \frac{t}{F_k} \right] \cdot Q_k. \tag{12}
\]

4 Comparing overrun and skipping

As described in [3], none of the synchronization mechanisms is superior to the others, because their performance heavily depends on the system parameters. The performance of the mechanisms is measured by the amount of CPU resource required at system level (globally) in order to guarantee the schedulability of the subsystems and their internal tasks. The mechanism that requires the lowest CPU resource is considered as the best among all mechanisms. One way to compare the performance of the mechanisms is by comparing the interfaces of the subsystems using each mechanism. Although this comparison can be useful at subsystem level, it is not useful at the system level, i.e., when subsystems are integrated, each of the mechanisms schedule subsystems globally in a different way. In this section, we therefore compare the mechanisms using the notion of system load [5] since it provides an indication of the system CPU requirement in the presence of shared resources. The comparison is done by means of simulation experiments. We start this section by briefly recapitulating the notion of system load. Next, we describe the setup of our experiments, which is followed by the results.

4.1 System load

For comparison purposes, system load is defined as a quantitative measure to represent the minimum amount of CPU allocations necessary to guarantee the global schedulability of the system \( S \). System load \( \text{load}_{\text{sys}} \) is calculated as follows:

\[
\text{load}_{\text{sys}} = \max \{ \alpha_s \} \tag{14}
\]

where

\[
\alpha_s = \min_{0 < t \leq P_s} \left\{ \frac{\text{RBF}_s(t) + B_s}{t} \mid \text{RBF}_s(t) + B_s \leq t \right\}. \tag{15}
\]

Note that \( \alpha_s \) is the smallest fraction of the CPU resources that is required to schedule a subsystem \( S_s \) (satisfying Eq. (3)) assuming that the global resource supply function is \( \alpha_s t \).

Given a system consisting of more than one subsystem, each subsystem containing a set of tasks, an efficient synchronization mechanism is the one that produces the lowest system load \( \text{load}_{\text{sys}} \) for this system. If \( \text{load}_{\text{sys}} > 1 \) then the system is not schedulable.
4.2 Experiment definition

The simulation study is performed by applying the synchronization mechanisms ONP, OWP, and SKP on 1000 different randomly generated systems using the schedulability analysis presented in section 3.2, where each system consists of 5 subsystems and, in turn, each subsystem contains 8 tasks. Once a system is generated, it is analyzed by applying the three synchronization mechanisms. The internal resource ceiling of the globally shared resources is assumed to be equal to the highest task priority in each subsystem and we assume that $T_i = D_i$ for all tasks. 2-6 tasks access globally shared resources in each subsystem. The worst-case critical section execution time of a task $\tau_i$ is set to a value between $0.3C_i$ and $0.8C_i^4$. A task is assumed to access at most one globally shared resource. For each system, the number of the global shared resources is two and all subsystems have at least two internal tasks that access both global shared resources. For each simulation study the following settings are changed and a new 1000 systems is generated:

1. System utilization – the system utilization, i.e., the summation of the utilization of all tasks in the system, is specified to a desired value.
2. Subsystem period – the subsystem period is specified as a range with a lower and upper bound. The simulation program generates a subsystem period randomly within the specified range, following a uniform distribution.
3. Task period – the task period is specified and generated in the same way as the period of a subsystem. The system utilization is divided randomly among the subsystems and the assigned utilization to each subsystem is in turn divided randomly to the tasks that belong to that subsystem. Since the task period is generated to a value within the interval as specified, the execution time is derived from the desired task utilization. All randomized system parameters are generated following uniform distributions.

4.3 Simulation results

Tables 1-4 show the results of the 4 different simulation studies performed.

- **Study 1** is specified having a task utilization of 15%, task periods between 400 and 1000, and subsystem periods between 50 and 200.
- **Study 2** decrease task periods (compared to Study 1) to the interval of 400-450, and therefore also $X_s$ (a task will have the same utilization compare to Study 1 and since its period is lower then its execution time becomes lower).
- **Study 3** decrease subsystem periods (compared to Study 1) to the interval of 50-60.
- **Study 4** increase task utilization (compared to Study 1) to 30%, and therefore also $X_s$ (a task utilization will be higher and that increases its execution time).

In each of these studies, 1000 systems are randomly generated, and for each generated system all three synchronization mechanisms are applied. For each case, the load$_{sys}$ is calculated hence making it possible to determine which of the three mechanisms requires the lowest load$_{sys}$ for that particular system. Keeping track of all calculated load$_{sys}$ values allow for determining min, max and average of the set of 1000 systems. Also, looking at the 1000 systems, it is possible to determine how often a particular mechanism requires less CPU resource than the other (shown under "Best" row in Tables 1-3). For reference, the same calculation is performed without sharing any global resources as well, to indicate the cost of running the synchronization protocols, and it is shown in the tables under HSF.

### 4.3.1 Skipping vs. overrun

<table>
<thead>
<tr>
<th></th>
<th>ONP</th>
<th>OWP</th>
<th>SKP</th>
<th>HSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.48</td>
<td>0.48</td>
<td>0.42</td>
<td>0.3</td>
</tr>
<tr>
<td>Min</td>
<td>0.36</td>
<td>0.33</td>
<td>0.31</td>
<td>0.24</td>
</tr>
<tr>
<td>Max</td>
<td>0.70</td>
<td>0.85</td>
<td>0.61</td>
<td>0.46</td>
</tr>
<tr>
<td>Best</td>
<td>7%</td>
<td>3%</td>
<td>90%</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 1. Measured results of Study 1**

Looking at Table 1, the results from using SKP is relatively better than both other mechanisms. The reason for this is that the range of task periods is much higher than the range of subsystem periods. Decreasing the range of task periods and keeping the subsystem period range forces SKP to require a higher load$_{sys}$, as show in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>ONP</th>
<th>OWP</th>
<th>SKP</th>
<th>HSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.47</td>
<td>0.48</td>
<td>0.46</td>
<td>0.33</td>
</tr>
<tr>
<td>Min</td>
<td>0.36</td>
<td>0.33</td>
<td>0.32</td>
<td>0.24</td>
</tr>
<tr>
<td>Max</td>
<td>0.62</td>
<td>0.64</td>
<td>0.63</td>
<td>0.43</td>
</tr>
<tr>
<td>Best</td>
<td>40%</td>
<td>10%</td>
<td>50%</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 2. Measured results of Study 2**

On the other hand, decreasing the range of subsystem periods and keeping the range of task periods the same as in Study 1 makes the performance of SKP much better than the other two mechanisms, as shown in Table 3.

To explain the reason, comparing Eq. (4) and Eq. (8) (local schedulability analysis), $\tau_{\text{best}}(t)$ is higher when using
SKP compared to when using ONP and maybe OWP (inherent in the effect of self blocking that is added in Eq. (8)) which makes the minimum required budget higher when using SKP. However, the difference between the subsystem budget when using SKP and when using ONP and OWP depends on the following parameters; subsystem period $P_s$, task period $T_t$, critical section execution times of the shared resources $c_{i,j}$ and $X_s$ for OWP (see Eq. (6)). For SKP, if $T_t \gg P_s$, then a small increment in the subsystem budget may be enough to cover the effect of self blocking on the $rbf_i(t)$ (to satisfy the schedulability condition $rbf_i(t) = sbf(f(t)$ at $t = T_t$).

On the other hand, the effect of using ONP and OWP appears in the calculation of $RBF(t)$ (global schedulability) as the maximum overrun $X_s$, added to $RBF(t)$ (see Eq. (5) and Eq. (7)). Hence, if the difference between subsystem budget when using SKP, and subsystem budget when using either ONP or OWP, is much less than $X_s$, then SKP will require a lower $load_{syst}$ as the $load_{syst}$ depends on $RBF(t)$ (see Eq. (14) and Eq. (15)).

### Table 3. Measured results of Study 3

<table>
<thead>
<tr>
<th></th>
<th>ONP</th>
<th>OWP</th>
<th>SKP</th>
<th>HSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.58</td>
<td>0.74</td>
<td>0.39</td>
<td>0.37</td>
</tr>
<tr>
<td>Min</td>
<td>0.41</td>
<td>0.45</td>
<td>0.28</td>
<td>0.23</td>
</tr>
<tr>
<td>Max</td>
<td>0.80</td>
<td>&gt; 1</td>
<td>0.58</td>
<td>0.57</td>
</tr>
<tr>
<td>Best</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Schedulable</td>
<td>100%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 4. Measured results of Study 4

The results of Study 4 are outlined in Table 4. In Study 4 the system utilization is increased which in turn increases the values of the critical section execution time and that increases the value of $X_s$. In this study it is clear that SKP is performing better than the other two mechanisms. Following the same reasoning as above, if the maximum execution time inside a critical section becomes larger, and the range of task periods is much higher than the range of subsystem periods, then SKP will perform better than ONP and OWP.

4.3.3 Number of shared resources

One of the factors that can decrease the performance of SKP is the number of shared resources accessed by tasks that belong to a subsystem. Increasing the number of shared resources will increase the $rbf_i(t)$ of task $t_i$ since it sums the critical section execution times as shown in Eq. (9)-(11), while for the overrun mechanism, calculating $rjf_i(t)$ includes only the maximum blocking from lower priority task. We can conclude that SKP performs better if a low number of resources are accessed by tasks.

5 Summary

In this paper, a simulation study has been performed on two different synchronization mechanisms; the overrun mechanism and the skipping mechanism. These mechanisms are used to enable the sharing of global logical resources in a Hierarchical Scheduling Framework (HSF), and the study investigates their efficiency in terms of CPU resource requirements. The results from the simulation

---

The same conclusion was shown in [9] and it is presented in this paper to show that it is also valid with the scheduling framework that is used in this paper.
study show that skipping can perform better than the overrun mechanism if the task periods are much larger than their subsystem period. Otherwise and for a high difference between subsystems periods, the overrun with payback can give better results, and for equal or close to equal subsystem periods, the overrun without payback performs better. Future work include implementing and testing the mechanisms on real industrial systems. Also, it would be interesting to develop an approach that selects different protocols, on subsystem level, in order to minimize the overall amount of required CPU resources.

References