A new approach for global synchronization in hierarchical scheduled real-time systems

Citation for published version (APA):

Document status and date:
Published: 01/01/2009

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 02. Aug. 2019
A new approach for global synchronization in hierarchical scheduled real-time systems

Moris Behnam, Thomas Nolte
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

We present our ongoing work to improve an existing synchronization protocol SIRAP [4] for hierarchically scheduled real-time systems. A less pessimistic schedulability analysis is presented which can make the SIRAP protocol more efficient in terms of calculated CPU resource needs. In addition and for the same reason, an extended version of SIRAP is proposed, which decreases the interference from lower priority tasks. The new version of SIRAP has the potential to make the protocol more resource efficient than the original one.

1 Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical CPU sharing among applications under different scheduling services [17]. 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 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 course-grained subsystems. In the HSF a subsystem provides an introspective interface that specifies the timing properties of the subsystem precisely [17]. 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, 10], intra-subsystem resource sharing is addressed using existing synchronization protocols for resource sharing between tasks, e.g., the Stack Resource Policy (SRP) [2]. Recently, three SRP-based synchronization protocols for inter-subsystem resource sharing have been presented, i.e., HSRP [6], BROE [9], and SIRAP [4].

In this paper we focus on SIRAP, improving the associated schedulability analysis and making the protocol more efficient in terms of CPU resource usage. The contributions of this paper are twofold. Firstly, it removes some pessimism in the schedulability analysis of the SIRAP protocol. Secondly, this paper proposes a change in the original SIRAP protocol to reduce the interference from lower priority tasks which may reduce the required CPU resources that the subsystem needs to guarantee the schedulability of all its internal tasks.

2 Related work

Over the years, there has been a growing attention to hierarchical scheduling of real-time systems [1, 5, 7, 8, 10, 11, 12, 14, 16, 17]. Deng and Liu [7] proposed a two-level HSF for open systems, where subsystems may be developed and validated independently. Kuo and Li [10] presented schedulability analysis techniques for such a two-level framework with the Fixed-Priority Scheduling (FPS) global scheduler. Mok et al. [15, 8] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [17] introduced the periodic virtual processor model (to characterize the periodic CPU allocation behaviour), and many studies have been proposed on schedulability analysis with this model under FPS [1, 12, 5] and under EDF scheduling [17, 19]. However, a common assumption shared by all above studies is that tasks are independent.

Recently, three SRP-based synchronization protocols for inter-subsystem resource sharing have been presented, i.e., HSRP [6], BROE [9], and SIRAP [4]. An initial comparative assessment of these three synchronization protocols [3] revealed that none of them was superior to the others, however. In particular, the performance of the protocol turned out to be heavily dependent on system parameters.
3 System model and background

This paper focuses on scheduling of a single node or a single network link, where each node (or link) is modeled as a system $S$ consisting of one or more subsystems $S_s \in S$. The system is scheduled by a two-level HSF. During runtime, the system level scheduler (global scheduler) selects, at all times, which subsystem will access the common (shared) CPU resource.

Subsystem model A subsystem $S_s$ consists of a task set $T_s$ and a local scheduler. Once a subsystem is assigned the processor (CPU), its scheduler will select which of its tasks will be executed. Each subsystem $S_s$ is associated with a subsystem timing interface $S_s(P_s, Q_s, X_s)$, where $Q_s$ is the subsystem budget that the subsystem $S_s$ will receive every subsystem period $P_s$, and $X_s$ is the maximum time that a subsystem internal task may lock a shared resource. Finally, both the local scheduler of a subsystem $S_s$ as well as the global scheduler of the system $S$ is assumed to implement the FPS scheduling policy. Let $R_s$ be the set of global shared resources accessed by $S_s$.

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 each element $c_{i,j} \in \{c_{i,j}\}$ is a critical section execution time that represents a worst-case execution-time requirement 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 same assumption is made for the subsystems. The set of shared resources accessed by $\tau_i$ is denoted $\{R_j^i\}$. Let $hp(i)$ return the set of tasks with priorities higher than that of $\tau_i$ and $lp(i)$ return the set of tasks with priorities lower than that 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. The motivation for this assumption is that higher $P_s$ will require more CPU resources [18].

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. For each logical resource, 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.

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 ($rc_{ij}$) for local scheduling and an external resource ceiling ($RX_j$) for global scheduling. They are defined as $rc_{ij} = \max\{i|\tau_i \in T_s \text{ accesses } R_j\}$ and $RX_j = \max\{s|S_s \text{ accesses } R_j\}$.

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 applies for subsystems from a global scheduling point of view.

4 SIRAP

SIRAP is based on the skipping mechanism. The protocol can be used for independent development of subsystems and supports subsystem integration in the presence of globally shared logical resources. It uses a periodic resource model [17] to abstract the timing requirements of each subsystem. SIRAP uses the SRP protocol to synchronize the access to global shared resources in both local and global scheduling. SIRAP applies a skipping approach to prevent the budget expiration inside critical section problem. The mechanism works as follows; when a job wants to enter a critical section, it enters the critical section at the earliest instant such that it can complete the critical section execution before the subsystem budget expires. This can be achieved by checking the remaining budget before granting the access to globally shared resources; if there is sufficient remaining budget then the job enters the critical section, and otherwise the local scheduler delays the critical section entering of the job (i.e., the job blocks itself) until the next subsystem budget replenishment. In addition, it sets the subsystem ceiling equal to the internal resource ceiling of the resource that the self blocked job wanted to access, to prevent the execution of all tasks that have priorities less than or equal to the ceiling of the resource until the job releases the resource.

Local schedulability analysis The local schedulability analysis under FPS is as follows [2, 17]:

$$\forall \tau_i \exists t: 0 < t \leq D_i, \text{ rbf}_{fps}(i, t) \leq \text{sbf}(t),$$ (1)
where \( \text{sbf}_s(t) \) is the supply bound function based on the periodic resource model presented in [17] that computes the minimum possible CPU supply to \( S_s \) for every interval length \( t \), and \( \text{rbf}_{FP}(i, t) \) denotes the request bound function of a task \( \tau_i \). \( \text{sbf}_s(t) \) can be calculated as follows:

\[
\text{sbf}_s(t) = \begin{cases} 
  t - (k + 1)(P_s - Q_s) & \text{if } t \in V^{(k)} \\
  (k - 1)Q_s & \text{otherwise}, 
\end{cases} 
\tag{2}
\]

where \( k = \max \left( \left\lceil \left( t - (P_s - Q_s) \right)/P_s \right\rceil, 1 \right\rceil \) and \( V^{(k)} \) denotes an interval \([k + 1]P_s - 2Q_s, (k + 1)P_s - Q_s\).

Note that, for Eq. (1), \( t \) can be selected within a finite set of scheduling points [13]. The request bound function \( \text{rbf}_{FP}(i, t) \) of a task \( \tau_i \) is given by:

\[
\text{rbf}_{FP}(i, t) = C_i + I_S(i, t) + I_H(i, t) + I_L(i), 
\tag{3}
\]

\[
I_S(i) = \sum_{R_j \in \{ R_t^i \}} X_{i,k}, 
\tag{4}
\]

\[
I_H(i, t) = \sum_{\tau_j \in \text{lpb}(i)} \left( \frac{t}{T_j} \right) \left( C_j + \sum_{R_k \in \{ R_t^j \}} X_{j,k} \right), 
\tag{5}
\]

\[
I_L(i) = \max_{\tau_j \in \text{rpb}(i)} \left( 2 \cdot \max_{\forall R_j \mid r_{c_j} \geq t} \left( X_{f,j} + c_{f,j} \right) \right) \tag{6}
\]

where \( I_S(i) \) is the self blocking of task \( \tau_i \), \( I_H(i, t) \) is the interference from tasks with higher priority than \( \tau_i \), and \( I_L(i) \) is the interference from tasks with lower priority than \( \tau_i \), that access shared resources.

**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 lower CPU resources. 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 Eq. (1). Hence, \( Q_s = \text{calculateBudget}(S_s, P_s) \) (the function is similar to the one presented in [17]).

**Calculating \( X_s \)**

Any task \( \tau_i \) accessing a resource \( R_j \) can be preempted by tasks with priority higher than \( r_{c_j} \). Note that SIRAP prevents subsystem budget expiration inside a critical section of a global shared resource. This is achieved using the following equation:

\[
Q_s \geq X_s. 
\tag{7}
\]

From Eq. (7), \( X_s \leq Q_s < P_s \) 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 from the time that self blocking happens until the end of the next subsystem period. Then \( X_{i,j} \) can be computed as follows,

\[
X_{i,j} = c_{i,j} + \sum_{k=r_{c_j}+1}^{n} C_k, \tag{8}
\]

where \( n \) is the number of tasks within the subsystem. Let \( X_j = \max \{ X_{i,j} \} \) for all \( \tau_i \in T_j \) accessing resource \( R_j \), then \( X_s = \max \{ X_j \} \) for all \( R_j \in R_s \).

## 5 Improved SIRAP analysis

In this section we will show that Eq. (6) is pessimistic and can be improved such that the subsystem budget may decrease. Each task \( \tau_i \) that shares a global resource \( R_j \) with a lower priority task \( \tau_j \) can be blocked by \( \tau_j \) due to (i) self blocking of \( \tau_j \) and in addition due to (ii) access of \( R_j \) by \( \tau_j \). The maximum blocking times of (i) and (ii) are given by the self blocking time \( X_{f,j} \), and the maximum execution time \( c_{f,j} \) of \( \tau_j \) inside a critical section of \( R_j \), respectively. The worst-case blocking is the summation of the blocking from these two scenarios, as shown in Eq. (9).

\[
I_L(i) = \max_{\tau_j \in \text{rpb}(i)} \left( \max_{\forall R_j \mid r_{c_j} \geq t} \left( X_{f,j} + c_{f,j} \right) \right). \tag{9}
\]

Since \( c_{f,j} \leq X_{f,j} \), the interference \( I_L(i) \) of tasks with a priority lower than that of task \( \tau_i \), based on (8), is at most equal to that of (6). As a result, \( \text{rbf}_{FP}(i, t) \) may decrease, and the corresponding subsystem budget \( Q_s \) may therefore decrease as well.

## 6 The new approach

Looking at Eq. (1), one way to reduce the subsystem budget \( Q_s \) is by decreasing \( \text{rbf}_{FP}(i, t) \) for tasks that require highest subsystem budget. In Section 5, we have described one way to decrease \( \text{rbf}_{FP}(i, t) \) for higher priority tasks that share resources by decreasing \( I_L(i) \). In this section we propose a method that allows for a further reduction of \( I_L(i) \). According to SIRAP, when a task wants to enter a critical section it first checks if the remaining budget is enough to release the shared resource before the budget expiration. If there is not enough budget remaining, then the task blocks itself and changes only the subsystem ceiling to be equal to the ceiling of that resource. This prevents all higher priority tasks that will be released after the self blocking instance, and have priority less than or equal to the subsystem ceiling, from executing.

The new method is based on allowing all tasks with priority higher than that of the task that is in self blocking state to execute during the self blocking time. This can be achieved by setting the subsystem ceiling equal to the priority of the task that is in self blocking state (only in case of self blocking, and follow SRP otherwise). The main difference between SIRAP and the new approach is the setting of subsystem ceiling during self blocking. In SIRAP subsystem ceiling equals to the resource ceiling of the resource that cause the self blocking while using the new approach it will equal to the priority of the task that tried to access that resource. When using the new approach, the maximum interference from lower priority tasks \( I_L(i) \) will be decreased compared to Eq. (9), and can be calculated as;

\[
I_L(i) = \max_{\tau_j \in \text{rpb}(i)} \left( \max_{\forall R_j \mid r_{c_j} \geq t} \left( c_{f,j} \right) \right). \tag{10}
\]
According to the original SIRAP approach, if $\tau_i$ blocks itself, it should enter the critical section at the next subsystem budget replenishment. However, using the new approach there is no guarantee that $\tau_i$ will enter the critical section at the next subsystem activation, since tasks with priority higher than $\tau_i$ and less than the ceiling of $R_j$ are allowed to execute even in the next subsystem activation. To guarantee that $\tau_i$ will enter its critical sections at the next subsystem budget replenishment, the subsystem budget should be big enough to include the execution of those tasks. The following equation shows a sufficient condition to guarantee that there will be enough budget in the next subsystem activation to lock and release $R_j$ by $\tau_i$;

$$Q_s \geq X_{i,j} + \sum_{k \in \{i+1, \ldots, rc_j\}} C_k. \quad (11)$$

Since we assume that $2P_s \leq T_m$ then all higher priority tasks will be activated at most one time during the time $t \in [t_{rep}, t_{rep} + P_s]$ where $t_{rep}$ is the subsystem replenishment time after self blocking of task $\tau_i$.

Note that to evaluate $X_{i,j}$, Eq. (8) can be used without modification since the new approach changes SIRAP only within the self blocking time, and during the self blocking the task that cause self blocking is not allowed to access the shared resource.

Comparing Eq. (10) with Eq. (9), $I_L(i)$ may decrease significantly and that may decrease the subsystem budget. However, Eq. (11) adds a constraint which may require a higher subsystem budget. Given these opposite forces, we conclude that the new approach will not always decrease the minimum subsystem budget and therefore will not always give better results than the original SIRAP. We will illustrate this by the following example.

**Example:** Consider a subsystem $S_i$ that has three tasks and two of them share resource $R_1$ as shown in Table 1.

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$T_1$</th>
<th>$C_1$</th>
<th>$R_1$</th>
<th>$C_{i,j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>30</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>1</td>
<td>52</td>
<td>$R_1$</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>4</td>
<td>80</td>
<td>$R_1$</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 1. Example task set parameters**

Let the subsystem period be equal to $P_s = 15$. Using the original SIRAP, we derive $X_i = X_{i,1} = 6$ and $Q_s = 9.34$. Using the new approach, we derive $X_i = X_{i,1} = 6$ and $Q_s = 7$. This latter value satisfies Eq. (11), i.e., $Q_s \geq X_{i,1} + C_2 = 7$. In this case, the new approach decreases the subsystem budget, hence requires less CPU resources. Conversely, for $C_2 = 5$, we derive $Q_s = 10.67$ for the original SIRAP and derive $Q_s \geq X_{i,1} + C_2 = 11$ by applying Eq. (11) for the new approach. In this case, the original SIRAP outperforms the new approach.

**7 Summary**

In this paper, we have presented improved schedulability analysis for the synchronization protocol SIRAP and we have proposed a new approach which extends SIRAP. The improved analysis may decrease the minimum subsystem budget while still guaranteeing the schedulability of all tasks in a subsystem. The new approach has the same objective. The relative performance of the two versions of SIRAP strongly depends on the subsystem parameters as illustrated by means of an example. Hence, the original SIRAP is not superior to the new approach nor vice versa. Currently, we are developing an algorithm that selects for each task which approach (SIRAP or the new approach) that should be used to reduce the subsystem budget to a minimum.

**References**


