Improved SIRAP analysis for synchronization in hierarchical scheduled real-time systems

Citation for published version (APA):

DOI:
10.1109/ETFA.2009.5347234

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.
Improved SIRAP analysis for 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 on synchronization in hierarchical scheduled real-time systems, where tasks are scheduled using fixed-priority pre-emptive scheduling. In this paper, we show that the original local schedulability analysis of the synchronization protocol SIRAP [4] is very pessimistic when tasks of a subsystem access many global shared resources. The analysis therefore suggests that a subsystem requires more CPU resources than necessary. A new way to perform the schedulability analysis is presented which can make the SIRAP protocol more efficient in terms of calculated CPU resource needs.

1 Introduction

The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical CPU sharing among applications under different scheduling services [14]. 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, which may share (so-called global) logical resources requiring mutual exclusive access. 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. This means that subsystems can be independently developed, analyzed and tested, and later assembled without introducing unwanted temporal interference. Temporal isolation between subsystems is provided through budgets which are allocated to subsystems.

To ease the migration towards and integration of existing, legacy applications as subsystems into a HSF, fixed-priority pre-emptive scheduling (FPPS) must be supported for scheduling tasks of a subsystem, because FPPS is a de facto standard in industry today. Our current research efforts are directed towards the conception and realization of a two-level HSF that is based on (i) FPPS for both global scheduling of budgets (allocated to subsystems) and local scheduling of tasks (within a subsystem), (ii) the periodic resource model [14] for budgets, and (iii) the Stack Resource Protocol (SRP) [2] for both inter- and intra-subsystem resource sharing. The synchronization protocol for resource sharing, which is termed SIRAP [4], prevents depletion of a budget during global resource access through self-blocking of tasks, which may delay the access of a task to a global resource for at most one budget period.

In this paper, we re-consider the local schedulability analysis of our HSF. We show that the original analysis is very pessimistic when tasks of a subsystem access many global resources. The analysis therefore suggests that a subsystem requires more CPU resources than necessary. A new way to perform schedulability analysis is presented which can make our HSF more efficient in terms of calculated resource needs.

2 Related work

Over the years, there has been a growing attention to hierarchical scheduling of real-time systems. 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 an HSF assuming FPPS for global scheduling. Mok et al. [13, 8] proposed the bounded-delay virtual processor model to achieve a clean separation in a multi-level HSF. In addition, Shin and Lee [14] 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 FPPS [1, 11, 5] and under EDF scheduling [14, 16]. 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]. Unlike SIRAP, HSRP does not support local schedulability analysis of subsystems, and the local schedulability analysis presented for BROE in [9] is incomplete.

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 $S_s \in S$. The system is scheduled by a two-level HSF using FPPS at both levels.

Subsystem model A subsystem $S_s$ consists of a set $T_s$ of $n_s$ tasks and a FPPS-based 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. Let $R_s$ be the set of global shared resources accessed by $S_s$.

Task model This paper considers 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 within a critical section of a global shared resource $R_i$ (for simplicity of the presentation, we assume that each task access 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 $\{R_i\}$. 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 [15].

Shared resources 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 access to global shared resources, its associated terms resource, system and system 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 $RX_j = \max\{s|S_s\ \text{accesses}\ R_j\}$.

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 prevents depletion of CPU capacity during global resource access through self-blocking of tasks. When a job wants to enter a critical section, it first checks the remaining budget. If there is sufficient remaining budget to complete the critical section, then the job is granted entrance. 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 less priority 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 [14]:

$$\forall \tau_i \ \exists t : 0 < t \leq D_i, \ \text{sbf}_{FP}(i, t) \leq \text{sbf}_{s}(t), \ (1)$$

where $\text{sbf}_{s}(t)$ is the supply bound function based on the periodic resource model presented in [14] that computes the minimum possible CPU supply to $S_s$ for every interval length $t$, and $\text{sbf}_{FP}(i, t)$ denotes the request bound function of a task $\tau_i$. $\text{sbf}_{s}(t)$ can be calculated as follows:

$$\text{sbf}_{s}(t) = \begin{cases} t - (k + 1)(P_s - Q_s) & \text{if } t \in V^{(k)} \\ (k - 1)Q_s & \text{otherwise}, \end{cases} \ (2)$$

where $k = \max\left(\left\lfloor \frac{(t - (P_s - Q_s)/P_s)}{1}\right\rfloor, 1\right)$ and $V^{(k)}$ denotes an interval $\left[(k + 1)P_s - 2Q_s, (k + 1)P_s - Q_s\right]$.
Note that, for Eq. (1), \( t \) can be selected within a finite set of scheduling points [12]. The request bound function \( \text{rbf}_{FP}(i, t) \) of a task \( \tau_i \) is given by [4]:

\[
\text{rbf}_{FP}(i, t) = C_t + IS(i) + IH(i, t) + IL(i),
\]

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

\[
IS(i) = \sum_{R_k \in \mathcal{S}(R)} X_{i,k},
\]

\[
IH(i, t) = \sum_{\tau_h \in \mathcal{P}(r)} \left( \frac{t}{T_h} \right) \left( C_h + \sum_{R_k \in \{R^x\}} X_{h,k} \right),
\]

\[
IL(i) = \max_{\tau_j \in \mathcal{P}(i)} \max_{R_k \in \mathcal{P}(R)} \left( C_{j,k} + X_{i,k} \right).
\]

The term \( X_{j,k} \) in these latter equations represents the self-blocking of task \( \tau_j \) due to access to resource \( R_k \), and is equal to the maximum amount of capacity that the task (and tasks with a higher priority than \( rc_j \)) may need during resource access. Its value can be determined by

\[
X_{j,k} = c_{j,k} + \sum_{h=rc_k+1}^{n_s} C_h.
\]

**Subsystem budget** In this paper, it is assumed that the subsystem period \( P_s \) of a subsystem \( S_s \) is given while the minimum subsystem budget \( Q_s \) should be computed. We use \( \text{calculateBudget}(S_s, P_s) \) to denote a function that calculates this minimum budget \( Q_s \) satisfying Eq. (1). This function is similar to the one presented in [14].

Finally, when a task experiences self-blocking during a subsystem budget period it is guaranteed access to the resource during the next period. To provide this guarantee, the subsystem budget \( Q_s \) should be

\[
Q_s \geq X_s.
\]

### 5 Upper bound analysis

In this section we will show that the schedulability analysis associated with SIRAP is very pessimistic if many resources are accessed by tasks. We will show this by using the following example.

**Example:** Consider a subsystem \( S_2 \) that has three tasks as shown in Table 1.

Let the subsystem period be equal to \( P_s = 50 \). Using the original SIRAP analysis, we find a subsystem budget \( Q_s = 23.5 \). Task \( \tau_2 \) requires this budget in order to guarantee its schedulability (The set of points of time \( t \) used to determine schedulability of \( \tau_2 \) is \( \{100, 150\} \) and at time

\[
t = 150, \text{rbf}_{FP}(2, 150) = \text{rbf}_s(150) = 47.
\]

To evaluate \( \text{rbf}_{FP}(i, t) \) for \( \tau_i \), the SIRAP analysis assumes that the maximum number of self blocking instances will occur for \( \tau_i \) and all its lower and higher priority tasks. Considering our example, \( \text{rbf}_{FP}(2, 150) \) contains a total of 9 self blocking occurrences; 6 self blocking instances for task \( \tau_3 \) (see (5)), 2 for task \( \tau_2 \) (see (4)), and 1 for task \( \tau_1 \) (see (6)). Because \( \text{rbf}_{FP}(2, 150) = 47 \) and \( Q_s = 23.5 \), we know that \( \tau_2 \) needs at least two and at most three activations of the subsystem for its completion. Because no self-blocking instance can occur during a subsystem period in which a task completes its execution, the analysis should incorporate at most 2 self-blocking instances for \( \tau_2 \). This means that the SIRAP analysis adds 7 unnecessary self blocking occurrences when calculating \( \text{rbf}_{FP}(i, t) \) which makes the analysis pessimistic. If 2 self blocking occurrences are considered and the two largest self blocking values that may happen are selected (e.g., \( X_{3,2} = 2, X_{3,3} = 2 \)), then a subsystem budget of \( Q_s = 18.5 \) suffices. For this subsystem budget, we once again find at most 2 self-blocking instances. In other words, the required subsystem utilization can be decreased by 20% compared with the original SIRAP analysis.

In the following section we present a method to incorporate an upper bound on the number of self-blocking occurrences in \( \text{rbf}_{FP}(i, t) \) based on the length \( t \) of the interval.

### 6 Improved SIRAP analysis

The following conjecture presents an upper bound on the number of self blocking occurrences in an interval of length \( t \).

**Conjecture 1** An upper bound on the number of self blocking occurrences \( z(t) \) in an interval of length \( t \) is given by

\[
z(t) = \left\lfloor \frac{t}{P_s} \right\rfloor.
\]

After evaluating \( z(t) \), it is possible to calculate the self blocking \( IS(i, t) \) on task \( \tau_i \) from all tasks, i.e., lower priority tasks, higher priority tasks and the task \( \tau_i \) itself. Let's define \( G_i(t) \) as a multi-set that includes the following elements:

- \( \text{max}(X_{i,k}) \) for all \( \tau_l \in \mathcal{P}(i) \) and all \( R_k \in \{R^l\} \) for each \( \tau_l \), i.e., from lower priority tasks.
- \( X_{i,k} \) for all \( R_k \in \{R^l\} \), i.e., on all resources that are accessed by \( \tau_i \).

---

1 The equation for \( IL(i) \) is taken from [3].
• \( W_h(t) \times X_{h,k} \) for all \( \tau_h \in \text{hp}(i) \) (higher priority tasks) and all \( R_k \in \{ R^h \} \) for each \( \tau_h \), where \( W_h(t) = \lceil \frac{t}{\tau_h} \rceil \) is the maximum number of times that \( \tau_h \) activates within time \( t \). We mean by \( W_h(t) \times X_{h,k} \) the number of copies \( W_h(t) \) of \( X_{h,k} \) that will be inserted in \( G_i(t) \).

Note that the multi-set \( G_i(t) \) includes all \( X_{h,k} \) that may contribute to the self blocking, but only \( z(t) \) number of elements are required to be considered when evaluating the self blocking effect, and to consider the worst case scenario, the value of these elements should be the highest in the multi-set. In order to find the \( z(t) \) highest values, we define a sequence \( G^\text{sort}_i(t) \) that contains all elements of \( G_i(t) \) in a non-increasing order, i.e., \( G^\text{sort}_i(t) = \text{sort}(G_i(t)) \). Hence, the first element \( G^\text{sort}_i(t)[1] \) has the largest value. The self blocking effect from all tasks on \( \tau_i \) can now be determined by

\[
I^*_Z(i, t) = \sum_{j=1}^{z(t)} G^\text{sort}_i(t)[j].
\] (10)

Note that if \( z(t) \) is greater than the number of elements in the set \( G^\text{sort}_i(t) \), then we assume that the value of the extra elements are equal to zero.

We now replace \( I_H(i, t) \) and \( I_L(i) \) by \( I^*_H(i, t) \) and \( I^*_L(i) \), respectively, i.e., terms without self-blocking:

\[
I^*_H(i, t) = \sum_{\tau_h \in \text{hp}(i)} \left\lfloor \frac{t}{\tau_h} \right\rfloor C_h, \quad \text{(11)}
\]

\[
I^*_L(i) = \max_{\tau_l \in \text{hp}(i)} \left( \max_{R_k | v_c, a \geq 1} (c_{l,k}) \right). \quad \text{(12)}
\]

And finally \( \text{rbf}_{FP}(i, t) \) can be evaluated as

\[
\text{rbf}_{FP}(i, t) = C_i + I^*_Z(i, t) + I^*_H(i, t) + I^*_L(i). \quad \text{(13)}
\]

Returning to our example, we find \( z(150) = 3 \). Based on Eq (13), we find a minimum subsystem budget \( Q_s = 19.5 \), which is better than the original SIRAP. The analysis is still pessimistic, however, because \( z(t) \) is an upper bound on the number of self blocking occurrences rather than an exact number and in addition, \( t \) is selected from the schedulability test points set of \( \tau_2 \) rather than the Worst Case Response Time (WCRT) of the task. note that the WCRT of \( \tau_2 \) is less than 150 which adds more pessimism on the results.

7 Summary

In this paper, we have shown that the local schedulability analysis for the synchronization protocol SIRAP is pessimistic when many resources are accessed by the tasks of a subsystem. This pessimism is caused by the fact that the original analysis does not take into account that the maximum number of self blocking instances of tasks is bounded by the maximum number of subsystem period intervals in which these tasks execute. We presented a conjecture with an upper bound for the number of self blocking occurrences of tasks in an interval of length \( t \), and an improvement of the analysis of the SIRAP protocol based on that conjecture. Further improvements of the analysis, e.g., finding a tight upper bound on the number of self blocking occurrences, are a topic of future work.

References


