Transformation of Sporadic Tasks for Off-line Scheduling with Utilization and Response Time Trade-offs

Jens Theis and Gerhard Fohler
University of Kaiserslautern, Germany
{jtheis,fohler}@eit.uni-kl.de

Abstract

Many safety critical real-time systems follow the time triggered approach using off-line scheduling methods for reasons of determinism, simple fault tolerance, certifiability, etc. By their very nature, off-line schedules cannot handle activities which are not known completely at design time. While some methods have been presented to integrate event-triggered activities at runtime, these require modification of the simple and deterministic table look up scheduling at runtime.

In this paper, we provide a proactive method for handling sporadic tasks with standard off-line scheduling in time-triggered systems without changing the schedule at runtime. The method is based on transforming the parameters of sporadic tasks to create periodic reservation tasks, for including in regular task sets for standard off-line schedulers. The resulting scheduling tables can be executed by regular time-triggered systems. The transformation requires more reserved utilization than the sporadic tasks actually need. We use this not needed reserved utilization to reduce the worst case response time of the sporadic tasks. A trade-off between reserved utilization and worst case response time of sporadic tasks is presented in this paper, too. We provide a feasibility analysis also considering the switching overhead for given bounds on the reserved utilization and worst case response time.

In this paper, we provide a proactive method for handling sporadic tasks with standard off-line scheduling in time-triggered systems without changing the schedule at runtime. Our method is based on transforming the parameters of the sporadic tasks to create periodic reservation tasks, which can be scheduled by regular off-line schedulers. Thus, the sporadic tasks can be guaranteed off-line to meet their timing constraints, while their actual on-line arrival times remain unknown (but bounded by their inter-arrival time). We determine parameters for a transformation of sporadic tasks into periodic reservation tasks for inclusion into the off-line scheduling process. The reservation tasks are composed of so-called slices to determine the amount and location of reserved execution time. These reservation tasks guarantee that sporadic tasks meet their deadlines at runtime independent of their (unknown) arrival times.

1 Introduction

Safety critical real-time systems often follow the time-triggered approach using off-line scheduling methods [11] for reasons of determinism, simple fault tolerance, certifiability, etc. By their very nature, off-line schedules cannot handle activities which are not known completely at design time. While some methods have been presented to integrate event-triggered activities at runtime [8], these require modification of the simple and deterministic table look up scheduling at runtime.

Applications also demand sporadic tasks which react to events with unknown actual arrival times, but with minimum inter-arrival times between two consecutive releases. Because of these unknown arrival times of sporadic tasks, schedulers can only handle them at runtime [4]. Time-triggered schedulers can only schedule tasks with known release times.

When guaranteeing sporadic tasks at design time, there must be enough reserved execution time (RET) for the sporadic tasks in the off-line schedule. This has to be done for every time interval with the length of the sporadic task execution windows (time between release and deadline) within the schedule for every sporadic task in the system.

In this paper, we provide a proactive method for handling sporadic tasks with standard off-line scheduling in time-triggered systems without changing the schedule at runtime. Our method is based on transforming the parameters of the sporadic tasks to create periodic reservation tasks, which can be scheduled by regular off-line schedulers. Thus, the sporadic tasks can be guaranteed off-line to meet their timing constraints, while their actual on-line arrival times remain unknown (but bounded by their inter-arrival time). We determine parameters for a transformation of sporadic tasks into periodic reservation tasks for inclusion into the off-line scheduling process. The reservation tasks are composed of so-called slices to determine the amount and location of reserved execution time. These reservation tasks guarantee that sporadic tasks meet their deadlines at runtime independent of their (unknown) arrival times.

A given off-line scheduler (e.g. [15, 7, 1]) schedules the sporadic tasks off-line as periodic reservation tasks. One benefit of this proactive approach is the easy certifiability in contrast to reactive on-line methods. As opposed to reactive methods, our method includes the sporadic activities in the creation of the off-line schedule and is thus not relying on sufficient resources “left over” in a given off-line scheduling table. Periodic and sporadic tasks are guaranteed off-line, so
that they meet their deadlines. Reservation tasks are adapted to a trade-off between reservation task’s utilization and worst case response time (WCRT). We provide a feasibility test for given bounds on the utilization of the reservation tasks and the WCRT. Furthermore, we consider the switching overhead and determine feasibility bounds of the reservation tasks including this overhead.

Mok transformed sporadic tasks into periodic tasks based on the slack of the sporadic tasks [14]. His transformation restricts the instances of the transformed tasks to start execution at the beginning of their execution windows. Based on off-line scheduling, Isović and Fohler used the slot-shifting approach [8] to handle sporadic tasks [9]. This approach is reactive in the sense that it takes the off-line schedule as given and attempts to use the remaining time for sporadic tasks. It also requires changes to the basic table look up scheduling of time-triggered systems. Šuča and Hanzálek [18] developed a non-preemptive scheduling algorithm for tasks with timing constraints and start time related deadlines. They used branch and bound and integer linear programming solutions. Furthermore, there are server based scheduling algorithms for event-triggered systems which handle sporadic tasks at runtime. The Total Bandwidth Server [16] and the Constant Bandwidth Server [2] are examples of such servers. Lipari and Bini showed in [12] an analysis to determine server parameters in order to provide CPU reservation for a set of periodic and/or sporadic tasks. They used optimization techniques to find parameters for minimum server utilization accounting for switching overheads and temporal guarantees. The server parameters considered for the optimization must satisfy a sufficient schedulability test, but the period-budget-pairs for minimum utilization may be outside this feasibility area. Their solution characterized the behavior of the server using the $(\alpha, \Delta)$-model proposed by Mok [6] and converted back to the period-budget-model. Our approach solves the problem only within the period-budget-domain, hence, providing for clarity and simplicity of the solution. We can also provide a trade-off between the WCRT of the sporadic tasks and the reserved utilization accounting for switching overheads. In [5], Davis and Burns showed an analysis for server algorithms with fixed priorities. In [10], Jeffay and Stone presented conditions to solve feasibility and schedulability problems in dynamic priority task systems.

The remainder of the paper is organized as follows: section 2 gives an overview over used terms, symbols and definitions. Section 3 depicts our method with the assumptions made and a proof of the concept. The consequences on the utilization of the reservation tasks and the WCRT of the sporadic tasks can be seen in section 4 and in section 5, respectively. Section 6 presents the trade-off between reservation tasks’ utilization and WCRT. Furthermore, it shows the feasibility analysis. After this trade-off section, an example illustrates the transformation and results in section 7. In section 8, we draw conclusions.

2 Terms, Symbols and Definitions

In the following, we refer to relative time values (e.g. worst case execution times) by using upper case variables, whereas lower case variables represent absolute time values (e.g. release times).

2.1 Tasks

The set $P = \{\tau_P^1, ..., \tau_P^n\}$ represents $n$ periodic tasks $\tau_P^i$ with possibly complex time constraints, such as end-to-end demands. Periodic tasks are composed of a sequence of an infinite number of consecutive instances $\tau_P^{i,\omega}$ with $\omega \in \{1, ..., \infty\}$. These instances are released periodically. Periodic tasks’ parameters are: worst case execution time (WCET) $C_P^i$, period $T_P^i$ and relative deadline $D_P^i$. The instances are released at $r_P^{i,\omega} = (\omega - 1)T_P^i$ and have absolute deadlines at $d_P^{i,\omega} = r_P^{i,\omega} + D_P^i$. Hence, periodic tasks are characterized by the tuple $\tau_P^i = (C_P^i, T_P^i, D_P^i)$.

Furthermore, the set $S$ represents $m$ sporadic tasks $\tau_S^i$. Sporadic tasks [14] are composed of a sequence of an infinite number of consecutive instances $\tau_S^{i,\omega}$ with $\omega \in \{1, ..., \infty\}$. They have arbitrary release times with a minimum inter-arrival time $T_S^i$. Further parameters of the sporadic tasks are the WCET and deadline. Hence, sporadic tasks are characterized by the tuple $\tau_S^i = (C_S^i, T_S^i, D_S^i)$.

The WCRT $R_S^i$ of a sporadic task $\tau_S^i$ determines the longest possible time interval between release and successful execution of a task instance [13].

The properties of the transformed sporadic tasks are defined in section 3 when we show the transformation method of the sporadic tasks.

2.2 Schedule

We do not assume any specific off-line scheduler. The only restriction is that the scheduler can create a feasible schedule for the given task set. The transformed sporadic tasks – which we call reservation tasks – are included into the off-line scheduling process.

Equation (1) calculates the utilization of one periodic task with implicit deadlines.

$$U_P^i = \frac{C_P^i}{T_P^i}$$

The maximum workload of sporadic tasks is produced when they arrive with maximum frequency and they behave like periodic tasks [3]. For our transformation method, we assume this maximum workload scenario and the period (minimum inter-arrival time)
is equal to the deadline. For these assumptions, we will guarantee the execution of the sporadic tasks. When sporadic tasks arrive periodically with minimum inter-arrival time and the deadline is equal to the minimum inter-arrival time then the utilization of a sporadic task is defined as shown in equation (2).

\[ U^S_i = \frac{C^S_i}{T^S_i} \]  

(2)

The utilization of the entire schedule is determined by the sum of all utilizations.

\[ U = \sum_{i=1}^{n} U^P_i + \sum_{i=1}^{m} U^S_i \]  

(3)

3 Transformation of Sporadic Tasks into Reservation Tasks

We transform each sporadic task \( \tau^S_i \) into a periodic reservation task \( \tau^R_i \). As a consequence, the set \( R \) contains \( m \) reservation tasks.

3.1 Worst Case Arrival of Sporadic Tasks

We assume deadlines of the sporadic tasks as equal to the minimum inter-arrival time. For sporadic tasks with shorter deadlines, because we transform the sporadic tasks based on their execution window, the transformation can be adapted by using the deadline instead of the minimum inter-arrival time for the calculation of the reservation task parameters. Guaranteeing sporadic tasks with \( D^S_i \leq T^S_i \) will result in more reserved utilization than for sporadic tasks with \( D^S_i = T^S_i \).

Within the execution window of every sporadic task \( \tau^S_i \in S \), we reserve at least \( C^S_i \) (split into \( k_{\tau_i} \) slices). A slice of a reservation task is the amount of execution time reserved within one period of the reservation task. The sum of \( k_{\tau_i} \) slices is equal to the WCET of the sporadic task. This reservation is to be done for every possible release time of a sporadic task instance within the schedule. If we can guarantee a sporadic task instance for its worst case arrival then we can also guarantee it in all other cases.

Figure 1 shows worst case arrival: a sporadic task instance of \( \tau^S_i \) arrives directly after the scheduled reserved execution time (RET) \( C^R_i \). This RET before the release of the sporadic task instance is scheduled at the beginning of its period and thus, the time until the start of the next period \( (T^R_i - C^R_i) \) is maximum. This time until the next period elapses without execution of the sporadic task instance. All further slices are scheduled at the end of their periods so that the execution of the sporadic task instance completes just before its deadline. For the execution of the sporadic task instance, there are \( k_{\tau_i} \) slices and hence, \( k_{\tau_i} \) periods plus the time interval \( T^R_i - C^R_i \) without any execution needed – see equation (5a).

\[ C^R_i = \frac{C^S_i}{k_{\tau_i}} \text{ with } k_{\tau_i} \in \mathbb{N} \setminus \{0\} \]  

(4)

\[ (T^R_i - C^R_i) + k_{\tau_i} \cdot T^R_i = T^S_i \]  

(5a)

\[ (k_{\tau_i} + 1) T^R_i - C^R_i = T^S_i \]

\[ T^R_i = \frac{T^S_i + C^R_i}{k_{\tau_i} + 1} \]

(5b)

As a consequence, the resulting utilization of the reservation task is:

\[ U^R_i = \frac{C^R_i}{T^R_i} = \frac{C^S_i k_{\tau_i} + C^S_i}{T^S_i k_{\tau_i} + C^S_i} = \frac{U^S_i k_{\tau_i} + U^S_i}{k_{\tau_i} + U^S_i} \]  

(6)

As equation (6) shows, the utilization of a reservation task depends only on the utilization of the sporadic task and \( k_{\tau_i} \). With increasing \( k_{\tau_i} \), we can achieve a finer granularity and hence, we can decrease the needed utilization to guarantee sporadic tasks (see Figure 2).
The extreme points of the reservation task’s utilization are calculated in equations (7) and (8). Depending on \( k_{r_t} \), the reservation task’s utilization could be decreased to the sporadic task utilization if the WCET of the sporadic task is divided into an infinite number of slices.

\[
U_i^R(k_{r_t} = 1) = \frac{2U_i^S}{1 + U_i^S} \quad (7)
\]

\[
\lim_{k_{r_t} \to \infty} U_i^R = U_i^S \quad (8)
\]

![Figure 2. Utilization of a reservation task](image)

But an infinite number of slices for one sporadic task instance execution leads to an infinite number of preemptions of the sporadic task instance. We show the influence of this switching overhead in section 4. Each sporadic task has its own reservation task. Hence, we guarantee each sporadic task independently from the other sporadic tasks so that they do not interfere.

3.3 Theorem and Proof

In the following, we show that independent of the release time of a sporadic task instance, this method reserves enough execution time, so that the sporadic task instance can complete its execution within its execution window.

**Theorem 1:**

In each time interval \([t_1; t_2]\) with \( t_2 - t_1 \geq T^S \), i.e. arrival \( r^S_{i, \omega} \geq t_1 \) and deadline \( d^S_{i, \omega} \leq t_2 \), if \( C^S_i \) is the execution demand of the sporadic task instance (with relative deadline \( D^S_i \)), there will be at least \( C^S_i \) time units reserved for the execution of the sporadic task instance.

**Proof 1:**

To prove the theorem, we will show that there are at least \( C^S_i \) time units reserved within the interval \([t_1; t_2]\):

\[
\left[ \frac{t_2 - t_1}{T^R_i} \right] C^R_i \geq C^S_i
\]

Replacing \( C^R_i \) by its definition:

\[
\left[ \frac{t_2 - t_1}{T^R_i} \right] C^S_i \geq C^S_i
\]

Multiplying with \( \frac{k_{r_t}}{C^S_i} \):

\[
\left[ \frac{t_2 - t_1}{T^R_i} \right] \geq k_{r_t}
\]

Hence, the interval \([t_1; t_2]\) must be greater than or equal to the product \( k_{r_t} \cdot T^R_i \):

\[ t_2 - t_1 \geq k_{r_t} T^R_i \]

Replacing \( T^R_i \) by its definition:

\[ t_2 - t_1 \geq k_{r_t} \frac{T^S + C^S_i}{k_{r_t} + 1} \]

\[ \iff t_2 - t_1 \geq k_{r_t} T^S + C^S_i \]

We perform polynomial division for the right side of the inequality:

\[ (k_{r_t} T^S + C^S_i) : (k_{r_t} + 1) = T^S + \frac{C^S_i - T^S}{k_{r_t} + 1} \]

Thus, the inequality results in:

\[ t_2 - t_1 \geq T^S + \frac{C^S_i - T^S}{k_{r_t} + 1} \]

\[ \iff T^S - C^S_i \geq T^S - (t_2 - t_1) \]

With \((t_2 - t_1) \geq T^S \iff T^S - (t_2 - t_1) \leq 0\) follows:

\[ \frac{T^S - C^S_i}{k_{r_t} + 1} \geq 0 \quad (9) \]

According to the definition of \( k_{r_t} \), equation (9) is greater than or equal to zero if the numerator of the fraction is greater or equal than zero. The numerator is greater or equal than zero if the WCET does not exceed the period which is a trivial condition. Hence, we proved the assertion.

4 Utilization of the Reservation Tasks

As mentioned before, we can change the needed utilization of the reservation tasks by modifying \( k_{r_t} \). In theory, we can divide the reservation tasks into an infinite number of infinitesimally short slices. But in practice, this is not possible because of the switching overhead (e.g., [17]). As a consequence, the time consumed by this switching has to be included into the feasibility tests.

Besides the switching overhead, the length of a clock cycle is also a restricting factor for the minimum length of a slice.
4.1 Overhead of Guaranteeing Sporadic Tasks

The reservation tasks are periodic and hence, their utilization can be calculated with equation (1). But besides the RET of the reservation tasks, we have to consider the switching overhead $\lambda$ which is added to the RET. For the guarantee of the sporadic tasks, we assume the worst case for the switching overhead, i.e. for each instance of the reservation task the maximum switching overhead $\lambda$ is needed. The resulting utilization $U_i^R$ is determined in equation (10).

$$U_i^R = \frac{C_i^R + \lambda}{T_i^R} = \frac{(C_i^S + k_{\tau_i} \lambda)(k_{\tau_i} + 1)}{k_{\tau_i} T_i^S + C_i^S}$$ (10)

If we set $\lambda = 0$ we obtain the utilization mentioned earlier without overhead consideration (see equation (6)). Figure 3 shows the comparison between reservation task’s utilization with and without overhead for an example sporadic task. Without considering the switching overhead, the utilization of the reservation task approximates the utilization of the sporadic task ($k_{\tau_i} \rightarrow \infty$). But in practice, the switching overhead is not negligible and we choose an integer value for $k_{\tau_i} \rightarrow \infty$ close to the minimum of the dashed function.

As Figure 4 shows, we reserve more utilization than actually needed even if the sporadic tasks arrive with maximum frequency. In the following, we calculate the amount of unused reserved utilization $\Lambda_i$. For this calculation, we consider the arrival pattern which creates the highest workload, i.e. the sporadic task instances always occur with their minimum inter-arrival time. Equation (11b) calculates the amount of unused reserved utilization $\Lambda_i$ of one sporadic task. We include the switching overhead into the reservation tasks so that $\Lambda_i$ accounts for the switching overhead and the overhead by unused slices. The reserved utilization includes the switching overhead and the difference to the actual execution of the sporadic task results in $\Lambda_i$ (see equation (11a)).

$$\Lambda_i = \frac{C_i^R}{T_i^R} - \frac{C_i^S}{T_i^S}$$ (11a)

$$\Lambda_i = \frac{k_{\tau_i} \lambda + C_i^S - \frac{(C_i^S)^2}{T_i^S}}{k_{\tau_i} T_i^S + C_i^S}$$ (11b)

Figure 5 shows the unused reserved utilization for the example task of Figure 4. For values of $k_{\tau_i}$ with $\Lambda_i(k_{\tau_i}) > \min(\Lambda_i(k_{\tau_i}))$, the switching overhead has a stronger influence on the utilization of the reservation task than the utilization reduction achieved by increasing $k_{\tau_i}$.

5 Worst Case Response Time

When choosing $k_{\tau_i}$ for a desired utilization of the reservation, we must not forget that reservation tasks have to be schedulable. In section 5 (equation (15)), we show the necessary schedulability condition for the reservation tasks.

As Figure 4 shows, we reserve more utilization than actually needed even if the sporadic tasks arrive with maximum frequency. In the following, we calculate the amount of unused reserved utilization $\Lambda_i$. For this calculation, we consider the arrival pattern which creates the highest workload, i.e. the sporadic task instances always occur with their minimum inter-arrival time. Equation (11b) calculates the amount of unused reserved utilization $\Lambda_i$ of one sporadic task. We include the switching overhead into the reservation tasks so that $\Lambda_i$ accounts for the switching overhead and the overhead by unused slices. The reserved utilization includes the switching overhead and the difference to the actual execution of the sporadic task results in $\Lambda_i$ (see equation (11a)).
periods and shorter slices, we achieve a finer granularity and can reduce the WCRT of the sporadic task instances. As a consequence, we adapt equations (4) and (5b), so that the periods and reserved execution times are shortened.

5.1 WCRT Reduction

In the following, we shorten the period and the length of the slices of the reservation tasks to achieve a finer granularity, but we do not want to change the utilization of the reservation tasks. As a consequence, we introduce a new variable \( k_{R_i} \) to reduce the WCRT \( R_i^S \) of the sporadic tasks. In contrast to \( k_{\tau_1} \), that changes the utilization of the reservation task, \( k_{R_i} \) only changes the granularity of the reservation tasks and hence, the WCRT of the sporadic task.

The original periods \( T_i^{R(k)} \) – see equation (5b) – and the length of the original slices \( C_i^{R(k)} \) – see equation (4) – of the reservation task \( \tau_i \) are split into \( k_{R_i} \) fractions (with \( k_{R_i} \in \mathbb{N}\setminus\{0\} \)). As Figure 6 shows, the utilization of the reservation tasks is unchanged (ignoring the constant switching overhead) but the slices are more regularly distributed over the schedule.

\[
\tau_i^{R(k)} = \begin{cases} \tau_i (k_{R_i} - 1) & k_{R_i} = 1 \\ \tau_i (k_{R_i} - j) & k_{R_i} = 2 \end{cases}
\]

Figure 6. Shortening period and length of slices of a reservation task

As a consequence, the equations calculating the period and reserved execution time (slice length) of the reservation tasks have to be adapted to these changes. Equations (12) and (13) show the adapted equations for RET calculation and for period calculation, respectively.

\[
C_i^{R(k)} = \frac{C_i^{R(k)}}{k_{R_i}} = \frac{C_i^{S}}{k_{R_i} k_{\tau_1}} \quad \text{with} \quad k_{R_i} \in \mathbb{N}\setminus\{0\} \quad (12)
\]

\[
T_i^{R(k)} = \frac{T_i^{R(k)}}{k_{R_i}} = \frac{T_i^{S} + C_i^{S}}{k_{R_i} (k_{\tau_1} + 1)} \quad (13)
\]

Thus, the utilization of the reservations tasks can be adapted with \( k_{\tau_1} \) and the WCRT can be adapted with \( k_{R_i} \).

An improvement of the WCRT will lead to an increasing utilization because of the switching overhead. Equation (14) shows the utilization of a reservation task including the adaptation for WCRT reduction.

\[
U_i^{R} = \frac{C_i^{R(k)} + \lambda}{T_i^{R(k)}} = \frac{C_i^{S} (k_{\tau_1} + 1) + \lambda k_{\tau_1} k_{R_i} (k_{\tau_1} + 1)}{T_i^{S} k_{\tau_1} + C_i^{S}} \quad (14)
\]

The utilization of the reservation tasks restricts the schedulability of the entire task set. This results in the necessary schedulability condition shown in equation (15).

\[
\sum_{i=1}^{n} U_i^{P} + \sum_{i=1}^{m} U_i^{R} \leq 1 \quad (15)
\]

The utilization of the periodic tasks is given by the designer and cannot be influenced. With \( k_{\tau_1} \) and \( k_{R_i} \), we can influence the utilization of the reservation tasks. Figure 7 shows the utilization of one example reservation task versus the adaptation factors \( k_{\tau_1} \) and \( k_{R_i} \). The utilization available for the guarantee of a specific sporadic task restricts the combinations of \( k_{\tau_1} \) and \( k_{R_i} \).

Figure 7. Utilization of one reservation task

Figure 8 depicts an example which shows that WCRT can be reduced when splitting the period and RET into shorter slices. Both calculations of reservation task’s parameters are based on the same sporadic task \( \tau_1 (C_1^{S} = 10, T_1^{S} = D_1^{S} = 70) \). The switching overhead is shown disproportionately large to show how it influences the WCRT.

For \( (k_{\tau_1} = 1, k_{R_i} = 1) \), the WCRT is 70 and with \( (k_{\tau_1} = 1, k_{R_i} = 2) \) the WCRT can be decreased to 55, in this example.

5.2 Results of WCRT Reduction

We assumed for the determination of the WCRT, that the sporadic task instance arrives at the end of a slice so that there is just enough time left for fetching the data into the cache (switching overhead \( \lambda \)). This slice is scheduled at the beginning of its period and the following slices are scheduled at the end of their
achieved for (k_k = 1). For chosen pairs of k we can again confirm the feasibility of our method.

We need to check that the WCRT is always shorter than the deadline, which is always possible. A trade-off between both parameters for a given bound of the reservation tasks’ utilization is also possible. We show a feasibility analysis for this trade-off between the utilization of the reservation tasks and the WCRT of the sporadic tasks in subsection 6.1.

### 6.1 Feasibility Analysis

In the following, we describe a feasibility analysis to determine pairs of k_k and k_k for given bounds on the WCRT and the utilization of the reservation tasks. As an example, we use one sporadic task τ_i with the parameters C_i^S = 10 and T_i^R = D_i^R = 70. Figure 10 shows a contour plot for the feasibility analysis with switching overhead set to 1% of the WCET of the task (λ = 0.1). Dashed lines depict the utilization of the reservation task τ_i^R. The maximum available utilization for this reservation restricts the possible pairs of k_k and k_k. Solid lines show the WCRT of the sporadic task. The curves represent contour lines with the same values of that variable.

To check the feasibility of given bounds, we determine the area which is surrounded by the contour lines for the given bounds. If there is no pair of k_k and k_k, the transformation is infeasible for the given bounds.

### 7 Example

In this section, we present an example: first showing that without considering the sporadic tasks already in the off-line scheduling process, deadline
Figure 9. WCRT of $\tau^S_1(C^S_1 = 10, T^S_1 = 70)$: 3D plot (left) and contour plot (right)

misses can occur. After this, we perform a feasibility analysis for given bounds on the utilization of the reservation tasks and the WCRT to determine feasible pairs of $k_{\tau_i}$ and $k_{R_i}$ for the reservation tasks. To conclude the example, we show that including the reservation tasks into the off-line scheduling process, deadlines are met.

In this example, we use a task set with three periodic tasks and two sporadic tasks (see Table 1). In a first step, the off-line scheduler scheduled the periodic tasks and at runtime the two sporadic tasks arrive at the beginning of the schedule. Figure 11 shows the occurring deadline miss of the sporadic task $\tau^S_2$.

Table 1. Example task set

<table>
<thead>
<tr>
<th>task $i$</th>
<th>$C_i$</th>
<th>$T_i = D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau^P_1$</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>$\tau^P_2$</td>
<td>20</td>
<td>95</td>
</tr>
<tr>
<td>$\tau^P_3$</td>
<td>30</td>
<td>150</td>
</tr>
<tr>
<td>$\tau^S_1$</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>$\tau^S_2$</td>
<td>20</td>
<td>110</td>
</tr>
</tbody>
</table>

In the following, we show how we can determine the parameters $k_{\tau_i}$ and $k_{R_i}$ for given bounds on the utilization and the WCRT. The utilization of the periodic tasks is about 58%. We want to use the remaining 42% to guarantee the sporadic tasks. We test the feasibility of the bounds $U^R_1 \leq 0.16$ and $R^S_1 \leq 100$ for $\tau^S_1$ and $U^R_2 \leq 0.26$ and $R^S_2 \leq 95$ for $\tau^S_2$. Figure 12 shows the contour lines for the given bounds. The areas with feasible pairs of $k_{\tau_i}$ and $k_{R_i}$ are highlighted. We choose the pairs of $k_{\tau_i}$ and $k_{R_i}$ so that the number of slices for each task is minimum. As a result, we choose $(k_{\tau_1}, k_{R_1}) = (2, 1)$ and $(k_{\tau_2}, k_{R_2}) = (2, 2)$ (marked in Fig. 12).

With the chosen parameters we transform the sporadic tasks into periodic reservation tasks and schedule them together with the periodic tasks $\tau^P_1$, $\tau^P_2$, and $\tau^P_3$ again. The resulting parameters for the reservation tasks are: $\tau^R_1(C^R_1 = 5, T^R_1 = 35)$ and $\tau^R_2(C^R_2 = 5, T^R_2 = 20)$. The utilizations of the reservation tasks ($U^R_1 \approx 0.143, U^R_2 = 0.250$) are below the desired bounds. Figure 13 (upper part) shows the resulting off-line schedule. Again, both sporadic tasks are released at the beginning of the schedule. Both tasks meet their deadline using the reserved execution time of their allotted reservation task.

8 Conclusion

Many safety critical real-time systems follow the time-triggered approach using off-line scheduling methods for reasons of determinism, simple fault tolerance, certifiability, etc. In this paper, we addressed
the issue of creating an off-line schedule without specific off-line scheduler for periodic tasks and including sporadic tasks into the off-line scheduling process to guarantee them. Hence, there is no need to change the schedule at runtime. Besides the guarantee of the sporadic tasks, we also considered the trade-off between resource over-provisioning and WCRT improvement.

Our solution is based on transforming the parameters of sporadic tasks to create periodic reservation tasks, which can be included in regular task sets for standard off-line schedulers. We introduced a variable \( k_\tau \in \mathbb{N} \) to split the WCET of the sporadic task into \( k_\tau \) slices. With \( k_\tau \), we calculated parameters of the reservation tasks and provided a strategy to improve the utilization of the reservation tasks so that the flexibility of the schedule is increased.

We also presented a method to reduce the WCRT of sporadic tasks: we introduced a second variable \( k_R \in \mathbb{N} \) to split the slices of the reservation tasks into shorter slices, thus, getting a finer granularity and hence, more flexibility.

The switching overhead has been taken into account for the calculations of the reservation tasks utilization. The utilization of reservation tasks and the switching overhead limit the choice of the reservation tasks parameters.

Analyzing the trade-off between the utilization of the reservation tasks and the WCRT of the sporadic task instances provided the possibility to match the needed utilization to guarantee the sporadic tasks and their WCRT with the available resources. Thus the method provides feasible combinations of the variables \( k_\tau \) and \( k_R \) for the reservation tasks parameters, which provides designer choice of parameters, in order to influence the reserved utilization and the WCRT. The result of the off-line scheduling process was a schedule combining periodic tasks and reservation tasks.

Future work will include feasibility and timing analysis for \( k_\tau, k_R \in \mathbb{R} \).

References

Figure 12. Feasibility analysis of $\tau_1^S$ and $\tau_2^S$.