Journal of Systems Architecture 66-67 (2016) 61–68

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Start time configuration for strictly periodic real-time task systems

Tianyu Zhang[a,*], Nan Guan[a], Qingxu Deng[a], Wang Yi[a,b]

[a] *Northeastern University, Shenyang, Liaoning, China*
[b] *Uppsala University, Uppsala, Sweden*

ARTICLE INFO

Article history:
Received 13 December 2015
Revised 4 April 2016
Accepted 25 April 2016
Available online 3 May 2016

Keywords:
Strictly periodic
Scheduling
Schedulability analysis
Real-time systems
Embedded systems

ABSTRACT

In many digital control systems, it is required to perform computation in a strictly periodic fashion to provide high control performance. System designers need to assign time slots that are infinitely repeated given a strict period for each task such that the time slots of different tasks do not overlap. While previous work has studied how to decide if a system is schedulable with a certain time slot assignment, it is still an unexplored area of how to select time slots for strictly periodic tasks to make them schedulable. In this paper, we propose an efficient method to solve the above problem. Our method explores the relations among task periods to improve the possibility of finding feasible start time configurations. Finally, we conduct experiments with randomly generated workload to evaluate the performance of the proposed method.

1. Introduction

Time-triggered scheduling is a well-suited approach for designing hard real-time systems. Time-triggered systems have great advantage over event-triggered systems of being easier to understand and analyze [6]. Furthermore, the time-triggered paradigm supports composability with respect to the temporal behavior of a system.

Strictly periodic scheduling [4] is a time-triggered method related to the strictly periodic tasks scheduling problem in which the time separating two successive executions of the same task is strictly equal to the associated period. The strictly periodic task model is motivated by the fact that control algorithms are usually designed under the assumption of a perfect periodic sampling and actuating model. Therefore, the recurring control tasks are required to execute in a strictly periodic manner to provide good control performance. The traditional periodic real-time task model, in contrast, will cause significant jitters in performing the control tasks and degrades the control performance [2], which may not be acceptable in many cases. In avionic systems, a single failure in a critical equipment can lead to catastrophic consequences. Thus, according to the ARINC 653 standard [1], the strict periodicity is a main constraint considered by scheduler in the Integrated Modular Avionics (IMA)[17].

To schedule a strictly periodic real-time task, we should set a start time for its first released instance, and thus the time slots for all its instances are known as they are repeated with a strict period. To schedule a strictly periodic task set, we should find a start time for each task such that the system is *schedulable*, namely, the execution of any pair two different tasks never overlap. In [12,13], it has been proved that the problem of assigning start times for a strictly periodic task set on multi-processor systems is NP-complete in the strong sense. And an alternative reduction from 3-partition can be constructed, showing that the problem remain NP-complete in the strong sense for the case that only one processor is available. In general it requires to enumerate all combinations of individual task start times, which is highly unscalable. Then, in a most recent work [9], authors studied and proposed a schedulability test not involving tasks start time parameters. This allows designers to check the feasibility of the strictly periodic tasks, i.e., there exists a scheduling table such that both strict periods and deadlines are met. However, in the design process of such systems, it is much more important to know how to select start times for tasks. At this point, in this paper, we develop efficient algorithms for suboptimal solutions on start time assignment for strictly periodic tasks. Our algorithm explores the relation between task's periods to improve the possibility to find feasible start time configurations. To the best of our knowledge, this is the first work to study efficient methods of start time selection for strictly periodic real-time tasks.

Related work. The schedulability research on periodic task model has been maturely studied [8,14], while the existing schedulability tests for periodic tasks are not suitable for strictly periodic tasks as strict-period is a more restricted constraint than a classical period. This results that it makes sense to study strictly

* Corresponding author. Tel.: +15744403691.
E-mail address: ztylll@126.com (T. Zhang).

http://dx.doi.org/10.1016/j.sysarc.2016.04.014
1383-7621/© 2016 Elsevier B.V. All rights reserved.

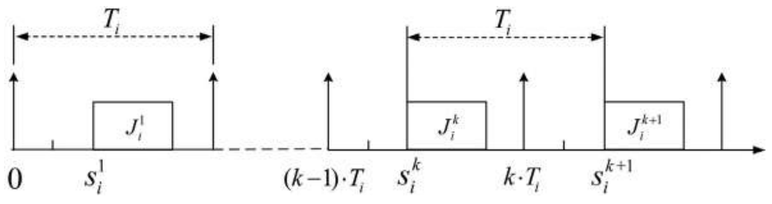


Fig. 1. Illustration of a strictly periodic task.

periodic tasks scheduling problem individually. In [12,13], Korst et al. showed that the problem of the non-preemptively scheduling strictly periodic tasks is NP-complete in the strong sense. Besides, they proposed a necessary and sufficient feasibility condition for a pair of tasks. In addition, Kermia and Sorel proposed in [10] a necessary schedulability condition which was proven to be very restrictive [15,16]. Eisenbrand et al. in [5] proposed a schedulability condition that requires all the periods of tasks in the system are harmonic, i.e., every period divides all other periods of greater value. In [15,16], Marouf and Sorel propoesd a similar work and gave a scheduling heuristic, however, based on the constraint that the period of new task has a multiple relationship with the existing task periods. In a most resent work [9], Kermia proposed a sufficient feasibility test established on a non-constructive proof for strictly periodic systems. More specifically, it only allows designers to check whether a correct scheduling table exists, but does not tell the designers the concrete start time configuration.

The remainder of this paper is organized as follows. In Section 2.1 we present the formal model that we use in this paper for representing strictly periodic real-time systems, define some important concepts and explain the runtime semantics of the system we seek to study. In Section 2.2 we introduce existing analysis on traditional strictly periodic task model including the exact analysis method we intend to compare with. In Section 3 we propose our start time selection method in this paper with a proof of its correctness and provide the complexity of it. In Section 4, we show some experiments to evaluate the performance of our proposed algorithm. Finally, conclusions are presented together with an indication of future work in Section 5.

2. Preliminaries

2.1. Model and definitions

In this section, we define the non-preemptive implicit deadline strictly periodic task system model studied in this paper. A task system τ consists of a set of independent recurring tasks. Each task $\tau_i \in \tau$ is characterized by a tuple $\langle T_i, C_i, s_i^1 \rangle$:

- T_i is τ_i's period.
- $C_i \in \mathbb{N}^+$ is the worst-case execution time of τ_i.
- $s_i^1 \in \mathbb{N}^+$ defines the start time instant of τ_i and $0 \le s_i^1 \le T_i$.

Each task τ_i releases potentially infinitely many *jobs*. We use J_i^k to denote the k^{th} job released by τ_i and s_i^k the start time of J_i^k. Note that

$$s_i^k = (k-1)\cdot T_i + s_i^1$$

For simplicity, we also use J_i to denote a job of τ_i and s_i to denote J_i's start time when the context is clear. Also we define the utilizations of task τ_i and task set τ as:

$$U_i = C_i/T_i, U_\tau = \sum_{\tau_i\in\tau} U_i$$

The non-preemptive strictly periodic system model has the following semantics. As shown in Fig. 1, the first job J_i^1 of task τ_i which released at 0 starts execution at time s_i^1, and executes for C_i without interruption (in a feasible schedule time slots assigned to different tasks do not overlap). Then, in every following period of τ_i, J_i^k released at r_i^k starts execution at time s_i^k and completes at $s_i^k + C_i$.

In this paper, we assume that time is discrete and clock ticks are indexed by natural numbers. Therefore, saying that a job J_i^k starts execution at time s_i^k means that it starts to be scheduled at the beginning of the interval $[s_i^k, s_i^k + 1)$. And all tasks are independent, that is, there are no precedence constraints among tasks. In addition, though we consider implicit deadline task model in this paper to present our new schedulability analysis method, our method can be easily modified to adapted for constrained and arbitrary deadline tasks.

In all, our aim is to present an efficient method finding a scheduling strategy for a strictly periodic task set τ. The scheduling strategy is established according to the knowledge of start time instant s_i^1 of each task τ_i assigned by the system designers using a priori timing analysis of the behavior of τ.

2.2. Existing analysis for strictly periodic task systems

In [12,13], Korst et al. proved that the problem of deciding whether a *correct scheduling strategy* for a given strictly periodic task set exists on a single processor is NP-complete in the strong sense. Besides, they proposed a necessary and sufficient schedulability test condition for a pair of tasks. This condition is presented in the following theorem.

Theorem 2.1. *[12,13] Two strictly periodic tasks τ_i and τ_j with given start times s_i^1 and s_j^1 can be correctly scheduled if and only if*

$$C_i \le (s_j^1 - s_i^1) \bmod gcd_{i,j} \le gcd_{i,j} - C_j \tag{1}$$

where $gcd_{i,j} = gcd(T_i, T_j)$.

The term $(s_j^1 - s_i^1) \bmod gcd_{i,j}$ captures the minimal distance from the execution of task τ_i to task τ_j. As shown in Fig. 2, this distance which represents the time duration that τ_j cannot interfere with τ_i's execution must be equal or larger than C_i. On the other hand, similarly, the right part of the inequation ensures τ_j will also not overlap with the execution of τ_i in the next time interval equals $gcd_{i,j}$. Note that, the rectangles in the figure only denote the conflict relation between these two tasks instead of the run-time executions. Specifically, assuming τ_i is configured with a start time s_i^1, time intervals with length of C_i start with an offset s_i^1 in each time interval equals $gcd_{i,j}$ cannot be used to execute τ_j. The ordering between τ_i and τ_j has no effect on condition (1) (as $-a \bmod b = (b-a) \bmod b$) and detailed proof of Theorem 2.1 can be found in [12].

Condition (1) can be generalized to more than two tasks by checking all pairs of tasks in the system with a sacrifice of complexity in some degree. When the start-time of each task is not given, to design a correct scheduling strategy, we can numerate all the possible start time instant configurations for the given task set.

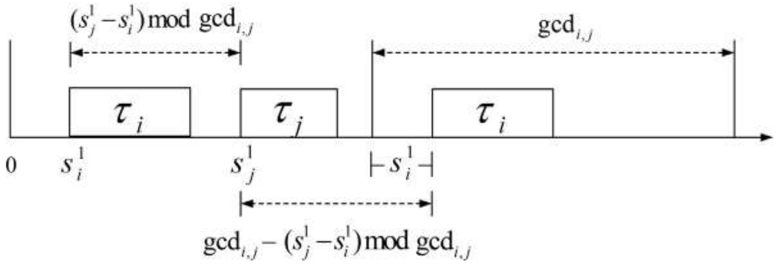

$(s_j^1 - s_i^1) \bmod gcd_{i,j}$
$gcd_{i,j}$
τ_i
τ_j
τ_i
0
s_i^1
s_j^1
s_i^1
$gcd_{i,j} - (s_j^1 - s_i^1) \bmod gcd_{i,j}$

Fig. 2. Explanation of Theorem 2.1.

That is, to solve the following set of linear inequalities:

$$\begin{cases} 0 \le s_i^1 \le T_i - C_i & \forall \tau_i \in \tau \\ C_i \le (s_j^1 - s_i^1) \mod gcd_{i,j} \le gcd_{i,j} - C_j & \forall \tau_i, \tau_j \in \tau \end{cases} \tag{2}$$

Condition (2) is an exact formulation of the schedulability test problem for strictly periodic systems. Any solution of the above inequalities can produce a correct schedule table for the given task set in which all tasks meet their strict periodicity constraints and deadlines. On the contrary, if the set of inequalities has no solution, this task set must be unschedulable.

It is easy to see that one can restrict to the start time in the range $0 \le s_i^1 \le T_i - C_i$. This results

$$\prod_{\tau_i \in \tau} (T_i - C_i + 1)$$

possible combinations for the task set. The number of combinations is exponentially increasing with the number of tasks and it is intractable to simply try all combinations.

To solve this problem, Kermia proposed in [9] a sufficient feasibility test established on a non-constructive proof not producing concrete tasks start-time parameters but only allowing designers to check whether a correct scheduling table exists. First, a so-called "Strict-Periodic Utilization Factor", denoted as $\mathbb{S}_i$, is proposed to represent the sum of time fractions utilized to execute τ_i at the same time the strict periodicity of tasks already scheduled are still met. $\mathbb{S}_i$ is computed as follows:

$$\mathbb{S}_i = \sum_{\tau_k \in \tau_{sched}} \frac{C_k}{gcd_{i,k}} + \frac{C_i}{T_i} \tag{3}$$

$\mathbb{S}_i$ is composed of two parts. The first part sums up $\frac{C_i}{gcd_{i,k}}$ for each task τ_k which has already been scheduled. This part excludes either time intervals used to execute all tasks scheduled or time intervals that τ_i cannot be used because of the possible conflict with those scheduled tasks. The second part concerns the factor $\frac{C_i}{T_i}$ seeing that C_i time units are required by τ_i in each period.

The schedulability test can be established based on this factor. The main idea is to pick a schedulable task (if any by checking $\mathbb{S}_i$) at each step from the original task set until all the tasks are identified as feasible. It reports a failure as soon as the test cannot find any task schedulable. At each step, it selects a "random" task as long as the selected task can pass the schedulability test and the decision made at each iteration is never reconsidered, which means there is no backtracking. This random task selection scheme is not effective and the overall approach is quite pessimistic. Moreover, their approach does not produce a concrete feasible start time configuration while it tells such a configuration indeed exists. Different from their work, our proposed method can give the feasible start time configuration.

3. Start time selection method

As we mentioned in the last section, the "random" task selection method in [9] can easily miss the opportunity of finding a feasible start time configuration. To solve the above problems, we propose a method which makes two-fold improvements: firstly, to design a new schedulability test algorithm which has a clear idea on checking the feasibility of a selected task; secondly, to propose a reasonable task selection method making our algorithm be more effective.

3.1. Schedulability test algorithm

To present the schedulability test algorithm, we first introduce a theorem as a sufficient test condition for a strictly periodic task τ_i with two definitions *conflict intervals* and *available intervals*.

Definition 3.1 (Conflict intervals)**.** *Conflict intervals* of task τ_i, denoted as CI_i, are intervals which cannot be used to execution τ_i in its first period. Otherwise, τ_i will conflict with tasks already in the scheduled set at some point.

Definition 3.2 (Available intervals)**.** *Available intervals* of task τ_i, denoted as AI_i, are intervals exclude the conflict intervals in its first period.

Let CI_i^k denote the conflict intervals of τ_i introduced by a task τ_k which has already scheduled in τ_i's first period. CI_i^k consists of two parts. The first part is the interval with a length of C_k starts from s_k^1. This interval is the execution of τ_k which will cause conflict immediately if we execute τ_i in this interval. The second part includes intervals with a length of C_k start from $s_k^1 + m \cdot gcd_{i,k}, m = 0, 1, \ldots$. These intervals also cannot be utilized to execute τ_i. Otherwise, the executions of τ_i and τ_k will must conflict with each other at some point in the future. Here we prove this property.

Proof. Assume the start instants of task τ_i and τ_k are s_i^1 and s_k^1 respectively. They start executing at time instant $t_i = s_i^1 + n_i \cdot T_i$ and $t_k = s_k^1 + n_k \cdot T_k$ in each of their periods, $n_i, n_k \in \mathbf{Z}$. As $T_i = n_1 \cdot gcd_{i,k}$ and $T_k = n_2 \cdot gcd_{i,k}$, $n_1, n_2 \in \mathbf{Z}$. We can get $t_i = s_i^1 + n_i \cdot (n_1 \cdot gcd_{i,k})$ and $t_k = s_k^1 + n_k \cdot (n_2 \cdot gcd_{i,k})$. Further, $t_i = s_i^1 + n_i' \cdot gcd_{i,k}$ and $t_k = s_k^1 + n_k' \cdot gcd_{i,k}$, $n_i', n_k' \in \mathbf{Z}$. If $s_i^1 = s_k^1 + m \cdot gcd_{i,k}$, then $t_i = t_k$ when $m = n_k' - n_i' \in \mathbf{Z}$. □

Then we get

$$CI_i^k = \bigcap_{m=0,\ldots,\frac{T_i}{gcd_{i,k}}} [s_k^1 + m \cdot gcd_{i,k}, s_k^1 + C_k + m \cdot gcd_{i,k}) \bigcap [0, T_i), \tag{4}$$

Example 3.3. Consider a task set $\tau = \{\tau_1, \tau_2\}$ with $(T_1, C_1) = (4, 1)$ and $(T_2, C_2) = (6, 1)$. τ_1 has already been scheduled with a start time $s_1^1 = 0$. Here we set $s_2^1 = s_1^1 + m \cdot gcd_{1,2}$ with an arbitrary integer $m = 3$. Then $s_2^1 = 0 + 3 \cdot 2 = 6$. This results that the executions of two tasks conflict at time instant 12 where it is the execution time both in the third period of τ_1 and in the second period of τ_2.

According to the definitions of conflict intervals and available intervals, we have

$$CI_i = \bigcap_{\tau_k \in \tau_{sched}} CI_i^k, \tag{5}$$

where τ_{sched} denotes the task set already scheduled and

$$AI_i = [0, T_i) - CI_i \tag{6}$$

Theorem 3.4. *A strictly periodic task τ_i is schedulable with a start time $s_i^1 \in [0, T_i - C_i]$ if it holds:*

$$[s_i^1, s_i^1 + C_i) \subseteq AI_i. \tag{7}$$

Proof. We prove the correctness of Theorem 3.4 by contradiction, assuming that τ_i satisfies condition (7) in Theorem 3.4 nevertheless τ_i is unschedulable with a start time s_i^1.

If task τ_i is not schedulable, this is necessarily due to:

1. either there is no time instant available for τ_i(i.e. existing s_i^1) to start when some tasks have already been scheduled before it.
2. or some time instants s_i^1 exist for starting τ_i, while there are no consecutive time intervals of length greater than or equal to C_i to execute τ_i.

We assume that tasks in $\{\tau_1, \dots, \tau_{i-1}\}$ have been scheduled before τ_i and those in $\{\tau_{i+1}, \dots, \tau_n\}$ will be tested after τ_i.

For the first case, there is no available time instant for τ_i to start means that all the time instants in $[0, T_i)$ are "occupied" by those tasks in $\{\tau_1, \dots, \tau_{i-1}\}$. Therefore, the time intervals required by task τ_k in $\{\tau_1, \dots, \tau_{i-1}\}$ without conflict with τ_i in $[0, T_i)$ is:

$$CI_i^k = \bigcap_{m=0,\dots,\frac{T_i}{gcd_{i,k}}} [s_k^1 + m \cdot gcd_{i,k}, s_k^1 + C_k + m \cdot gcd_{i,k}) \bigcap [0, T_i)$$

Thus, the total time intervals required by all the tasks in $\{\tau_1, \dots, \tau_{i-1}\}$ is

$$CI_i = \bigcap_{\tau_k \in \{\tau_1, \dots, \tau_{i-1}\}} CI_i^k.$$

Then, no available time instants exist implies that:

$$AI_i = \emptyset.$$

As τ_i satisfies condition (7) and $[s_i^1, s_i^1 + C_i)$ is not an empty set, this is in contradiction with the assumption.

For the second case, it indicates that length of each time interval in set AI_i is smaller than the required execution duration C_i of τ_i. This is also in contradiction with condition (7). In summary, both cases lead to contradictions, and so τ_i is schedulable if it holds the condition. □

The new schedulability test algorithm for strictly periodic task set τ, we name it STSP(τ), is shown in the pseudo-code of Algorithm 1. All tasks in τ are replicated in an initial set Γ_1 and the expected output of the algorithm is a schedulable task set Γ_2 contains all tasks in Γ_1 with assigned feasible start times.

As the first step, we compute g as the greatest common divisor of all the tasks in τ. Obviously, if $g = 1$, which means that there are at least two tasks with co-prime periods, we can easily deduce that this task set can never be schedulable. This conclusion can also be reached from condition (1). After ensuring the inexistence of tasks with co-prime periods, next we enter the while loop to search for a final schedulable task set. At each iteration, we select an appropriate task τ_i from Γ_1 and operate in following steps (MS(Γ_1) denotes the new task selection method which we will introduce in the next subsection). To schedule τ_i, we firstly check whether there is any time unit available for starting τ_i after some tasks have already been in the scheduled set. We consider all scheduled tasks in Γ_2 and analyze the execution behaviors of them. If AI_i is not empty, as the definition of AI_i, this indicates that there exists an available s_i^1 for τ_i. Next we will select and assign a feasible start time from those available ones. An available s_i^1 cannot determine whether a feasible start time exists because we need to check whether the time intervals beginning from each s_i^1 are enough for τ_i to execute uninterruptedly. To do this, we review AI_i to examine if there is any consecutive duration larger than or equal to C_i. If so, such a consecutive duration is legal for τ_i to execute and the beginning instant is set to τ_i's start time s_i^1. Notice, if more than one interval satisfy conditions above in AI_i, we always select the first appeared one. Because, on one hand, we have no idea about the following tasks' information to promote locating the optimal legal interval selection; on the other, our proposed task selection method can help us approaching to the optimal result. Passing all above conditions, τ_i can be safely picked into the schedulable task set Γ_2. If τ_i fails to pass any of the above conditions, the algorithm fails. Otherwise, it returns true with a schedulable task set Γ_2.

Algorithm 1 STSP(τ)
```
1: $\tau \leftarrow \{\tau_1, \cdots, \tau_n\}$
2: $\Gamma_1 \leftarrow \{\tau\}$, $\Gamma_2 \leftarrow \emptyset$
3: $g \leftarrow gcd_{1,\dots,n}$
4: if $g = 1$ then
5:     return FAILURE
6: end if
7: while $\Gamma_1 \neq \emptyset$ do
8:     $\tau_i \leftarrow$ MS($\Gamma_1$)
9:     $AI_i \leftarrow [0, T_i)$
10:    $s_i^1 \leftarrow -1$
11:    for $\tau_k \in \Gamma_2$ do
12:        $CI_i^k$ and $CI_i$ are computed by equation (4) and (5) respectively
13:        $AI_i \leftarrow AI_i - CI_i$
14:    end for
15:    if $AI_i = \emptyset$ then
16:        return FAILURE
17:    else
18:        for $t \in [0, \dots, T_i - C_i]$ do
19:            if $[t, t + C_i) \subseteq AI_i$ then
20:                $s_i^1 \leftarrow t$
21:            end if
22:        end for
23:    end if
24:    if $s_i^1 = -1$ then
25:        return FAILURE
26:    else
27:        $\Gamma_1 \leftarrow \Gamma_1 \setminus \{\tau_i\}$
28:        $\Gamma_2 \leftarrow \Gamma_2 \cup \{\tau_i\}$
29:    end if
30: end while
31: return $\Gamma_2$
```

Example 3.5. Here we illustrate the operation of Algorithm 1 especially the process of start time assignment for a certain task considering an example task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ such that $(T_1, C_1) = (4, 1)$, $(T_2, C_2) = (6, 1)$ and $(T_3, C_3) = (8, 1)$. We assume τ_1 and τ_2 both have scheduled with $s_1^1 = 0$ and $s_2^1 = 1$ respectively and now we check whether τ_3 can be put into the schedulable set.

To find a starting time-unit for τ_3 in its first period [0, 8), we firstly consider all conflict intervals caused by τ_1 and τ_2 in this period. Because the start time of both tasks are known, we can easily do so by using Eqs. (5) and (4) and get that $CI_3 = CI_3^1 \bigcap CI_3^2 =$

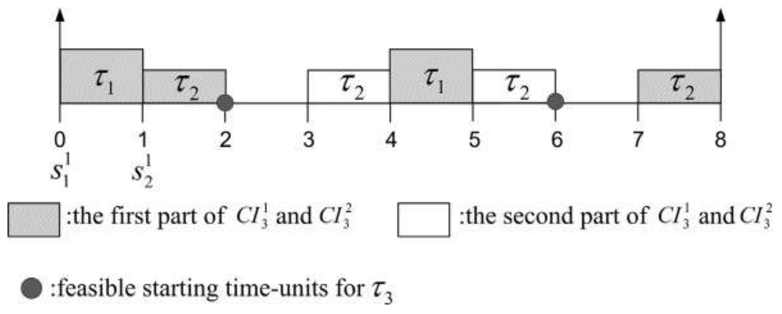

τ_1
τ_2
τ_2
τ_1
τ_2
τ_2
0 1 2 3 4 5 6 7 8
s_1^1
s_2^1
:the first part of CI_3^1 and CI_3^2
:the second part of CI_3^1 and CI_3^2
:feasible starting time-units for τ_3

Fig. 3. Start time instant assignment for τ_3.

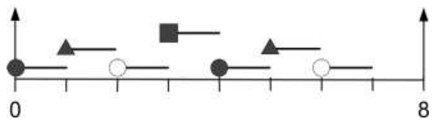


Fig. 4. Sequence $\{\tau_2, \tau_1, \tau_3\}$.

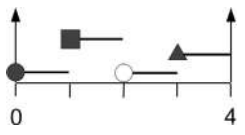


Fig. 5. Sequence $\{\tau_2, \tau_3, \tau_1\}$.

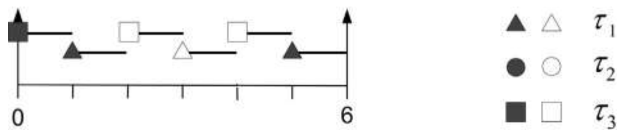


Fig. 6. Sequence $\{\tau_3, \tau_1, \tau_2\}$.

$\{[0, 2), [3, 6), [7, 8)\}$. As shown in Fig. 3, available intervals of τ_3 $AI_3 = \{[2, 3), [6, 7)\}$. Then feasible starting time-units are 2 and 6. While we cannot afford complexity caused by searching all the available intervals as depicted in this example. we just locate the first appeared one. As a result, we assign $s_3^1 = 2$ for τ_3. Finally, we can safely put τ_3 into the feasible task set Γ_2.

Note that, the proposed schedulability test algorithm is a sufficient but not a necessary condition mainly because of the following reasons. Firstly, feasibility decision made in each iteration of the while loop after a certain task selected is never reconsidered and this may miss the optimal result. Secondly, in the process of start time setting for a selected task, we directly choose the first suitable time unit in its first period because that we don't know information about all the unscheduled tasks at current point. Whether our proposed algorithm can find a feasible start time assignment, if there exists one, heavily depends on the order to select tasks. In the following, we present a task selection method that can improve the possibility for our algorithm to find feasible solutions.

3.2. Task selection method

In the last subsection, a missing part which we have not explained is the task selection method, named MS(τ) in algorithm STSP(τ) (line 8). Before going into the detailed introduction of MS(τ), we first discuss the role MS(τ) plays in the schedulability test. the following example shows that the task selection order results in different results for the same task set.

Example 3.6. Consider a strictly periodic task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ with $(T_1, C_1) = (4, 1)$, $(T_2, C_2) = (6, 1)$ and $(T_3, C_3) = (8, 1)$. We assume three different task selection sequences: $\{\tau_2, \tau_1, \tau_3\}$, $\{\tau_2, \tau_3, \tau_1\}$ and $\{\tau_3, \tau_1, \tau_2\}$. In any of these three sequences, the first two tasks are schedulable. Our point is to check if there exists any available start time instant for the last task. Figs. 4, 5 and 6 depict, respectively, the start time checking in each sequence. According to the formula of computing CI_i and AI_i:

$\{\tau_2, \tau_1, \tau_3\}$: $CI_3 = \{[0, 3), [4, 7)\}$ and $AI_3 = \{[3, 4), [7, 8)\}$;
$\{\tau_2, \tau_3, \tau_1\}$: $CI_1 = \{[0, 3)\}$ and $AI_1 = \{[3, 4)\}$;
$\{\tau_3, \tau_1, \tau_2\}$: $CI_2 = \{[0, 6)\}$ and $AI_2 = \emptyset$.

In the third case, strict periodicity of τ_2 cannot be met since all time instants are unavailable because of τ_1 and τ_3, which corresponds to the absence of solid circle symbols in Fig. 6. This indicates that if task τ_2 is the last one selected then this system is not feasible. However, this system is feasible in the first and the second cases in which the last task can be assigned feasible start instants and put into the final schedulable task set.

From the above example we see that different task selections do affect the final feasibility result for the same task set. This inspire us to design a better task selection method to improve the schedulability. Observed from Fig. 6, τ_2 fails to find an available time instant to start because that $gcd_{2,1}$ and $gcd_{2,3}$ are both small values (2). Then we have to reserve C_1 and C_3 time units in each $gcd_{2,1}$ and $gcd_{2,3}$ time interval respectively to avoid conflict. However, this is somewhat pessimistic. As the greatest common divisor $gcd_{1,3}$ of task τ_1 and τ_3 is a larger value, these two tasks can definitely "overlap" the above stated $gcd_{2,1}$ and $gcd_{2,3}$ time interval. This observation inspires us to explore the implication of $gcd_{i,j}$ to help designing the task selection method.

Definition 3.7 (Harmonic Chain $\mathcal{H}(p)$ [11]). A *harmonic chain* $\mathcal{H}(p)$ is a task set in which all the tasks are harmonic tasks with periods equal to integer multiple of p.

$\mathcal{H}(p)$ captures visualized and understandable informations about all the tasks' periods in it. That is, for all $\tau_i \in \mathcal{H}(p)$, $T_i = m \cdot p, m \in \mathbb{N}^+$. If arranging the tasks in an increasing order by their periods, $i < j \Leftrightarrow T_i \le T_j$, we can easily deduce that $\forall \tau_i \in \mathcal{H}(p), gcd_{1,i} = T_1 = p$ as T_1 is the smallest period. Correspondingly, $gcd_{i,j}$ between τ_i and any task τ_j with $T_j \ge T_i$ increases with i. Besides, an important observation in Example 3.6 is that, a small $gcd_{i,j}$ value is harmful to the schedulability of task τ_i scheduled after τ_j. Therefore, tasks can cause small *gcd* values should be processed prior at the time processor resource is still sufficient. Considering this observation and the property of *gcd* in a harmonic chain, we want value $gcd_{i,j}$ increasing in the processing of task selection to benefit the final schedulability. This leads to the first procedure in our task selection method MS(τ):

Procedure 1. MS(τ) selects the task with the smallest period in the same harmonic chain.

This procedure tells which task should be selected in a certain harmonic chain. While, without lose of generality, a strictly periodic task set τ may consist of tasks that can be classified into several harmonic chains. Then the problem is, how MS(τ) operates in a situation that more than one harmonic chains exist.

Assume that two harmonic chains, $\mathcal{H}(p), \mathcal{H}(q) \subseteq \tau\,(p \neq q)$, exist in the task set and there is a task $\tau_i \in \mathcal{H}(p)$. Note that, two harmonic chains may overlap with each other which means that the intersection of $\mathcal{H}(p)$ and $\mathcal{H}(q)$ is not null. Therefore, in this case, we group tasks in the intersection into the harmonic chain which has more tasks. For instance, consider four tasks with periods equal to 4, 6, 8 and 12. We let task with period 12 belong to the harmonic chain $\mathcal{H}(4)$ instead of $\mathcal{H}(6)$ though 12 is also integer multiple of 6. Here we assume $\mathcal{H}(p)$ contains more tasks than $\mathcal{H}(q)$ does. We can deduce that the *gcd* value between τ_i and any task in $\mathcal{H}(p)$ is much more likely larger than that between τ_i and any task in $\mathcal{H}(q)$. Again, considering the relation between *gcd* and schedulability, we come to the second procedure in MC(τ):

Procedure 2. MS(τ) selects the task in the harmonic chain in which the number of tasks is minimum.

Procedure 2 comes from the indication that tasks in a harmonic chain cannot avoid to suffer a possible small *gcd* value between tasks in another. So we rather suffer this at a beginning phase also when more processing resource is available.

Finally, our proposed task selection method MS(τ) executes according to the procedures stated above. It first partitions tasks into different harmonic chains. For each step it chooses the harmonic chain that contains the least tasks. In each harmonic chain, MS(τ) selects tasks in the increasing order with their periods. Now reconsider the task set in Example 3.6, according to the procedures in task selection method MS(τ) stated above, we come to the final feasible sequence $\{\tau_2, \tau_1, \tau_3\}$.

3.3. Complexity and efficiency

The proposed schedulability test algorithm has a pseudo-polynomial complexity of $O(n^2)$ and n is the number of tasks in strictly periodic task set. Notice, complexity of all *gcd* pre-computation in the processing of checking feasible start time instant should also be considered. In [3], the complexity of *gcd* can be improved to $O(m(\log m)^2 \log(\log m))$ for m-bit numbers. Moreover, authors proposed a quasilinear algorithm for computing all pairs *gcds* of n m-bit integers, based on fast integer arithmetic [7]. In a summary, our method can obtain results in a realistic time compared with the optimal algorithm which is extremely slow. This is also shown in our experiments in the next section. Such a result makes our method efficient in realistic applications which are usually composed of thousands of tasks.

4. Experimental evaluation

In this section we conduct experiments to evaluate the performance of our new schedulability analysis method. We compare to previous approaches including the exact method based on [12] and study the acceptance ratios of their corresponding schedulability tests.

The task systems used in the experiments are generated as follows. As we consider the general situation that each task set consists of both harmonic tasks and non-harmonic tasks, the generation of a random task is controlled by following parameters: the probability P^n of being a non-harmonic task; the maximum worst-case execution time C^{max}; the harmonic period set T^h and non-harmonic period set T^{nh}. Each new task τ_i is then generated as follows: C_i is drawn from the uniform distribution over {1, 2,..., C^{max}}. T_i is drawn from set $T^h = \{1500, 3000, 6000, 12000, 24000\}$ with probability $1 - P^n$, otherwise from set $T^{nh} = \{2^x \cdot 3^y \cdot 50, x \in [0, 4], y \in [0, 3]\}$ in which, as we know, are typical numbers of period lengths in real-world instances. A random strictly periodic task set is generated by starting with an empty task set $\tau = \emptyset$ and then random tasks will be successively added in. Specifically, each task set is generated according to a target utilization U^*. We allow the task set's utilization to fall within the small interval $[U^* - 0.005, U^* + 0.005]$ because it is difficult to get an exact utilization value with integer parameter tasks. Then random tasks are added into the task set successively until the utilization falls into the target range. If the utilization exceeds the target range, we delete the task and regenerate a new one.

As discussed in Section 2.2, the exact formulation based on Theorem 2.1 fails in converging for large problem instances within an acceptable time limit. When it is implemented in systems consist of tasks with different temporal parameters, the execution times may vary greatly. We have experimented 5 task instances and tasks are generated according to $P^n = 0.5, C^{max} = 500$. The corresponding execution times varied from about 15 minutes to 1 hour.

To evaluate our new analysis method by comparing to the exact method more efficiently, we control tasks' parameters within a relatively small scale. Particularly, we let $T^h = \{15, 30, 60, 120, 240\}$, $T^{nh} = \{2^x \cdot 3^y \cdot 5, x \in [0, 2], y \in [0, 2]\}$ and $C^{max} = 10$.

We compare the following schedulability analysis methods:

- **STSP**. The new schedulability test algorithm STSP(τ) presented in this paper.
- **Exact**. The exact schedulability method formulated in condition (2) based on Theorem 2.1 in [12,13].
- **First-Fit**. A first-fit heuristic based on Theorem 2.1 assigns start time instants for tasks in a first-fit fashion. Specifically, at each iteration, it finds a task can be assigned a start time satisfying condition (1) with all scheduled tasks and picks it into the scheduled set. First-Fit terminates with FAILURE if and only if there isn't any task meets the condition left.
- **RS**. A simplified STSP(τ) without using the task selection method MS(τ), i.e. implementing a Randomly Selection method.
- **Kermia**. The schedulability test proposed in [9].

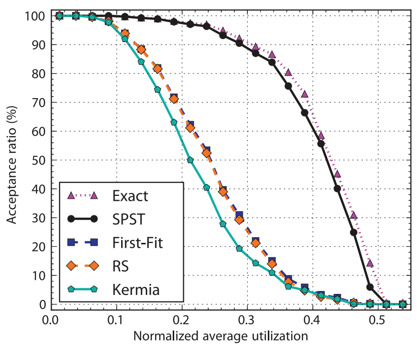
Fig. 7 depicts the acceptance ratio (fraction of schedulable task sets) as a function of (target) utilization for task sets generated using parameter $P^n = 0.1$ and $P^n = 0.5$ respectively. In general, a higher average utilization of target task set make a lower possibility to make the system schedulable. This can also be shown in the figure that each curve decreases with the utilization increasing. Each data point is based on 2000 randomly generated strictly periodic task sets. As shown in the figure, our new method STSP keeps an acceptable high performance comparing with the optimal exact method. Specifically, in the case of $P^n = 0.1$, the acceptance ratio of our proposed algorithm STSP is very close to that of the optimal algorithm Exact. On the other hand, there is a large gap between the acceptance ratios of the proposed approach in this paper and those of First-Fit, RS and Kermia.

Further, the average runtime execution time of the exact method and of our method STSP are evaluated with varying number of tasks in each task set. Fig. 8 shows that the running time of the exact method explodes quickly, while our method keeps a reasonable execution time when we vary the number between 20 and 80.

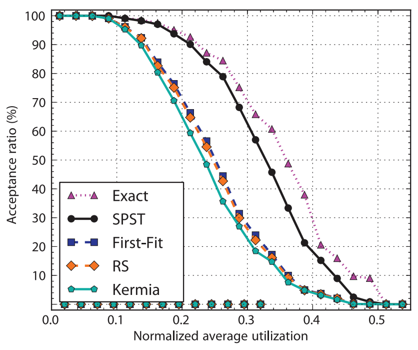
The above results indicate that our method gets a good tradeoff between schedulability and efficiency.

5. Conclusion

In this paper, we present a new schedulability analysis method for strictly periodic real-time systems. Our method provides



(a) $P^n = 0.1$



(b) $P^n = 0.5$

Fig. 7. Experiment results with different P^n for strictly periodic task sets.

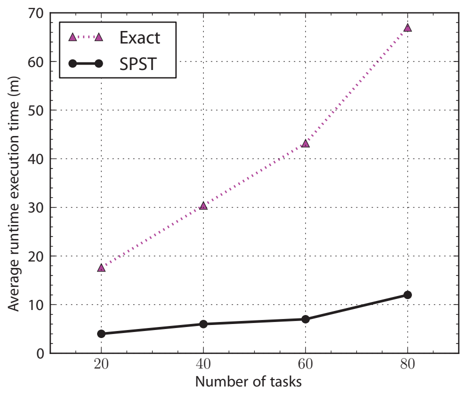


Fig. 8. Run-time overhead comparison.

a sufficient test condition to check the feasibility of a given task set, and returns the start time assignment if it is feasible. To improve the analysis effectiveness, we also present a reasonable task selection method. show that the new method indeed significantly and effectively improve the analysis precision.

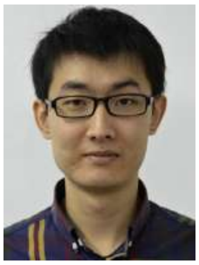
As the future work, we will further explore the start time configuration for strictly periodic systems. Such as, we will investigate how to assign feasible start time more effectively instead of directly choosing the first suitable time unit in this paper. In the special case that system consisting of tasks with unit-execution time, we intend to design an optimal method. We also plan to study how to apply the method in this paper to the partitioned multiprocessor scheduling problem. The challenge is to develop an efficient partitioning strategy considering the relation between task's periods.

Acknowledgment

This work is partially supported by China Scholarship Council, NSF of China under grant No.61472072, No.61528202 and National Key Basic Pre-Research Program of China (973 Program) under grant No.2014CB360509.

References

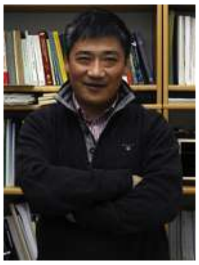
[1] ARINC, Avionics application software standard interface: ARINC specification 653-1, Technical Report, 2003.
[2] M. Behnam, D. Isovic, Real-time control design for flexible scheduling using jitter margin, Technical Report, Technical Report, Målardalen University, 2007.
[3] D.J. Bernstein, Fast multiplication and its applications,in: Buhler-Stevenhagen Algorithmic Number Theory book.
[4] L. Cucu, Y. Sorel, Schedulability condition for systems with precedence and periodicity constraints without preemption, in: Proceedings of 11th Real-Time Systems Conference, RTS, 2003.
[5] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, A. Wiese, Scheduling periodic tasks in a hard real-time environment, 37th International Colloquium on Automata, Languages and Programming (ICALP2010), 2010.
[6] W. Elmenreich, C. Paukovits, S. Pitzek, Automatic generation of schedules for time-triggered embedded transducer networks, ETFA, 2005.
[7] N. Heninger, Z. Durumeric, E. Wustrow, J.A. Halderman, Mining your ps and qs: detection of widespread weak keys in network devices, in: Proceedings of the 21st USENIX Conference on Security Symposium, 2012, p. 35.
[8] K. Jeffay, D.F. Stanat, C.U. Martel, On non-preemptive scheduling of period and sporadic tasks, RTSS, 1991.
[9] O. Kermia, Timing analysis of ttethernet traffic, J. Circuits, Systems, Comput. (2015).
[10] O. Kermia, Y. Sorel, Schedulability analysis for non-preemptive tasks under strict periodicity constraints, in: RTCSA, 2008, pp. 25–32.
[11] J.K. Kim, B.K. Kim, Probabilistic schedulability analysis of harmonic multi-task systems with dual-modular temporal redundancy, Real-Time Systems, 2004.
[12] J. Korst, Periodic multiprocessor scheduling, 1992 PhD thesis.
[13] J. Korst, E. Aarts, J.K. Lenstra, J. Wessels, Periodic multiprocessor scheduling, PARLE, 1991.
[14] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM, 1973.
[15] M. Marouf, Y. Sorel, Schedulability conditions for non-preemptive hard real–time tasks with strict period, RTNS, 2010.
[16] M. Marouf, Y. Sorel, Scheduling non-preemptive hard real-time tasks with strict periods, ETFA, 2011.
[17] A.A. Sheikh, O. Brun, P.-E. Hladik, B.J. Prabhu, Strictly periodic scheduling in ima-based architectures, Real-Time Syst. 48(4) (2012) 359–386.



Tianyu Zhang received his master's degree in computer applications technology from the Northeastern University (NEU), China in 2013. Currently, he is a PhD candidate at the institute of Cyber Physical Systems Engineering of the Northeastern University of China. His research interests are broadly in the area of real-time embedded systems with a focus on scheduling.

Nan Guan is a professor in the Institute of Embedded Systems, Northeastern University (NEU), China. He is also partially affiliated with the Embedded Systems Group chaired by Prof. Wang Yi at Uppsala University, Sweden. His research interests are in real-time embedded systems, especially real-time scheduling theory and worst-case execution time (WCET) analysis.

Qingxu Deng received his Ph.D. degree in computer science from Northeastern University, China, in 1997. He is a professor of the College of Information Science and Engineering, Northeastern University, China, where he serves as the Director of institute of Cyber Physical Systems Engineering. Starting from 2010, he is also the Director of the Key Lab. of Embedded Systems of Liaoning Province. His main research interests include Cyber-Physical systems, embedded systems, and Real-time systems.



Wang Yi is professor with a chair in Embedded Systems at Uppsala University. He received his Ph.D. in Computer Science in 1991 from Chalmers University of Technology. He is one of the initial contributors to the research area on verification of timed systems. As a co-founder of UPPAAL, a model checker for concurrent and real-time systems, he received the CAV 2013 Award with Kim Larsen and Paul Pettersson. His current interests include models, algorithms and software tools for modeling and verification, timing analysis, real-time scheduling, and their application to the design of embedded and real-time as well as mixed-criticality systems.