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/

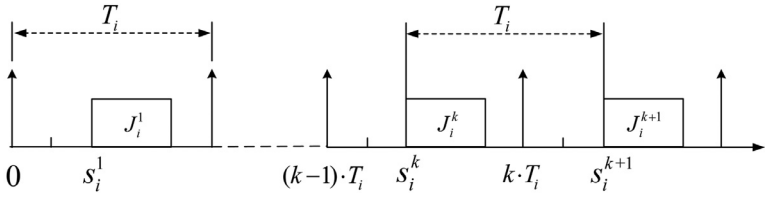

Fig. 1. Illustration of a strictly periodic task.

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*

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

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.