This work-in-progress paper was presented as part of the main technical program at IEEE ETFA'2011

Task period selection to minimize hyperperiod

Vicent Brocal[†] Patricia Balbastre[†] Rafael Ballester[‡]
Ismael Ripoll[†]
[†]Universitat Politécnica de Valencia, Spain
[‡]CFIS (Centre de Formació Interdisciplinària Superior), Universitat Politècnica de Catalunya

Abstract

In this paper a new task model with periods defined as ranges is proposed with the main goal of drastically reduce the hyperperiod of the task set. The model is focused to be applied in cyclic scheduling, where the length of the major cycle of the plan is determined by the hyperperiod. But it also can be applied in synthetic task sets generation, where having a small hyperperiod reduces complexity and simulation time. A new algorithm, which allows to calculate the minimum hyperperiod of such a set of tasks, is presented. This algorithm calculates the minimum value even with a large number of tasks, where exhaustive search becomes intractable.

1 Introduction

The periodic work model is the base of the real-time scheduling theory. This model has been shown to be practical and rigorous. It properly represents the fundamental timing behaviour of the real world, and also, it has been shown to be a formal model where rigorous theoretic analysis can be done. The system is described as a set of periodic tasks, which, in the basic model, consists of two parameters: C_i and P_i; the worst case execution time (WCET), and the period, respectively. Liu and Layland's seminal paper [5] established the basis of priority-driven schedulability analysis of periodic task models.

The periodic task model has been widely used because it copes with many application requirements. Nearly all of existing analysis theory assume that periods and computation times are fixed and known values. This assumption greatly simplifies the feasibility analysis, but limits the applicability of the results. Relaxing the classical assumptions on static task sets with fixed periods and deadlines can give higher resource utilisation, better control performance, energy saving, and can be used to adapt to dynamic environments, different operating conditions, overload situations, etc..

The basic model has been augmented to consider also deadlines shorter than periods, starting offsets, precedence, exclusive shared resources protocols, etc.

In this paper, we exploit the idea of period variation but with a different goal: to reduce the length of the hyperperiod. The hyperperiod H is calculated as the least common multiple (lcm) of the tasks' periods. Leung and Merrill [4] proved that the schedule produced by any preemptive scheduling policy over a periodic task set is cyclic, that is, a periodic real-time system repeats its arrival task pattern after an interval that coincides with the hyperperiod H. The first feasibility analysis consisted on checking the schedulability in the interval $[0, H)$ [4]. This test was improved later by Baruah et al [1] and by Ripoll et al [7] making the schedulability interval more accurate for EDF (earliest deadline first) scheduling algorithm.

In cyclic scheduling the hyperperiod is referred as the *major time frame*, which is the interval at which the entire schedule is repeated. The designer shall verify that the temporal constraints are met along the major time interval. Once the schedule is built, it is stored in a table which will be consulted on-line to select the active tasks. The shorter the hyperperiod the shorter the table, and consequently the smaller the memory footprint. For these two reasons, it is important to have a relatively small hyperperiod. In periodic systems with precedence constraints, the system of periodic tasks is usually described by a graph where dependent tasks are connected by precedence constraints. In order to reduce the complexity of the heuristic algorithm, the hyperperiod has to be a small value to reduce the number of operations in the dependence graph [3]. On the other hand, simulation experiments of real-time systems often require to have a bounded hyperperiod. That is, task load generators are implemented in such a way that a task set is dismissed when its hyperperiod exceeds a given limit. This is necessary to keep the simulation time within reasonable bounds. If task periods do not have common divisors, the hyperperiod can be a large value. In fact, as shown in [6], the hyperperiod grows exponentially with the greatest period and with the number of tasks.

For the above mentioned reasons, a task set with a small hyperperiod is a desirable feature. The most common technique is to select task periods to be harmonic, that is, all the periods have large common divisors. For example, in radar dwell applications, assigning for every task a harmonic period has the low overhead of maintain-

978-1-4577-0018-7/11/$26.00

ing constant temporal distance and schedulability analysis but is usually over-reserving the resources. To overcome this disadvantage, in [8] an algorithm is developed to transform task periods into synthetic ones, but they do not have to be harmonic. In partitioned systems, tasks of different partitions may have different time scales and different temporal requirements, therefore the usage of harmonic periods is not always possible.

1.1 Motivating example

Let's suppose three tasks whose periods are 20, 28 and 93. The hyperperiod is $H = \text{lcm}(20, 28, 93) = 13020$. The goal of this example is to show the reduction achieved in the hypeperdiod by means of a minimal modification of a task's period. In order to simplify the example, we will focus on the third task. If task period is given as a range, for example $[90, 95]$, then it is easy to see that $H = \text{lcm}(20, 28, 90) = 1260$. We have chosen the period inside the range that produces the smallest hyperperiod. This means that, although the desired period of the third task is 93, we admit a period between 90 and 95 if we benefit of an hyperperiod reduction.

By defining task periods as ranges of valid periods, it is possible to have a small hyperperiod by conveniently selecting the period from each task's range.

Let's extend the previous example so that all task periods are given as ranges, that is, the task set periods are $[17, 21], [25, 30], [90, 95]$. Which is the set of task periods that leads to the smallest hyperperiod? We would have to take into account all possible period combinations inside period ranges and compute the lcm for all of them.

Listing 1 shows how to calculate the minimum hyperperiod (H_{min}) of a set of periods ranges by checking all the possible combinations. For large task sets or wide period ranges, the cost of this algorithm may be unaffordable in most practical applications.

1.2 Contributions and outline

In this paper we propose an extension of the basic periodic model where the periods of the tasks are defined as a range of valid periods. We also present an efficient method for finding the set of periods from a given set of ranges that produces the minimum hyperperiod. The result of the proposed algorithm is a task set with fixed periods (the classical task model) which has the minimum hyperperiod.

The proposed algorithm can be used as a preprocessing step in any application where a long hyperperiod is not desirable.

The rest of the paper is organised as follows: Section 2 presents the problem to be solved and the system model. In section 3 the algorithm to find the minimum hyperperiod is presented. Finally, we summarise our conclusions in section 4.

```
1
2  Hmin:integer:=integer'last;
3  L,L':integer;
4  function ComputeLCM(L,i) is
5    if (i>n) then
6      if (L<Hmin) then
7        Hmin:=L;
8      end if;
9      return;
10   end if;
11   for x in li..ui loop
12     L'=LCM(L,x);
13     ComputeLCM(L',i+1)
14   end loop;
15 end ComputeLCM;
16
17 function MinHyperExhaustive(τ) is
18   ComputeLCM(1,1);
19 end MinHyperExhaustive;
```

Listing 1. Exhaustive search algorithm

2 System Model and Assumptions

Let $\tau = \{\tau_1, ..., \tau_n\}$ be a periodic task system with n tasks. The period of each task $\tau_i \in \tau$ is defined as a range $T_i = [l_i, u_i]$ of integers. Note that a range of periods does not mean that the task have a variable execution rate, but that the scheduler can select any value from this range. Once the period is selected, it is fixed and honored for all the schedule. That is, each task will be executed always at the same rate.

Definition 1 *Let* $\mathbb{H} = \{lcm(t_1, \cdots, t_n)\}$, *where* $t_i \in [l_i, u_i]$.

$\mathbb{H}$ *is the set of integers that are the lcm of all possible combinations of valid tasks periods.*

In what follows let $h \in \mathbb{H}$.

2.1 Problem statement

The problem we want to solve is to find $H_{min} = \min(\mathbb{H})$.

The minimum element of the set $\mathbb{H}$ can be found by calculating the set $\mathbb{H}$, sorting it and then selecting the first element. However, the cardinality of $\mathbb{H}$, given by $|\mathbb{H}| = \prod_{i=1}^{n} (u_i - l_i + 1)$, may be a quite large number. Listing 1 shows the implementation of the algorithm that calculates the set $\mathbb{H}$ and obtains the minimum element H_{min}.

For a large number of tasks, or wide periods ranges, the problem of finding H_{min} quickly becomes intractable.

2.2 Complexity of the problem

The cost of finding H_{min} is bounded by $O(O(lcm) \cdot |\mathbb{H}|)$, that is, the cost of the computing the lcm of n integers multiplied by the cardinality of $\mathbb{H}$.

The problem of computing the least common multiple reduces to the problem of computing the greatest common divisor (gcd). In this sense the gcd problem is analogous to the integer factorization problem, which has no known polynomial-time algorithm, but is not known to be NP-complete.

In our case, the problem of finding the minimum hyperperiod of a set of range of periods can be also reduced to the integer factorization problem. Let's suppose two tasks $\{\tau_1, \tau_2\}$ with range periods: $T_1 \in [x, x]$ and $T_2 \in [2, \sqrt{x}]$. Let's suppose that exists T_2 such that $x = aT_2$, being $a \in \mathbb{N}$. Then, $H_{min} = x$ and the problem of finding H_{min} has been reduced in this case to the problem of integer factorization (in particular, the problem of factorize the integer x).

The complexity of the integer factorization is not exactly known. The tightest asymptotic bound known for b bits number is:

$$O\left(exp\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}} log(b)^{\frac{2}{3}}\right)\right)$$

There are published algorithms that are faster than $O((1+\epsilon)^b)$ for all positive , i.e., subexponential time [2].

3 Fast hyperperiod search method

We propose an heuristic, the Fast Hyperperiod Search algorithm (FHS) that efficiently calculates the minimum hyperperiod of a set of ranges. It is important to note that the proposed algorithm calculates the exact solution rather than an approximated one. We call it heuristic because, there are few cases in which the algortihm takes infinite time in finding the solution. In these situations, the minimum hyperperiod is a huge value (greater than a lon long integer). The algorithm, when detects this situation, stops and returns a valid hyperperiod although not the minimum.

Definition 2 *Let g be the least common multiple of integers $(t_1, .., t_m)$, where t_i is a valid period for task τ_i and $1 \leq i \leq m < n$.*

$$g = lcm(t_1, \ldots, t_m) \ / \ t_i \in [l_i, u_i], 1 \leq i \leq m < n$$

Definition 3 *Let $\mathbb{G}$ be the set of all possible least common multiples for m tasks, that is, all possible values of g for m tasks.*

$$\mathbb{G} = \{lcm(t_1, \ldots, t_m) \mid \forall t_i \in [l_i, u_i], 1 \leq i \leq m < n\}$$

Initially, the algorithm pre-computes a set $\mathbb{G}$ containing **all** the valid hyperperiods for a subset of m tasks. This is the purpose of the `EnumerateLcm()` function used in Listing 2, though it is not explicitly shown.

Since each g is the *least common multiple* of periods for a subset of tasks,

```
1  function FHS (τ, m)
2  begin
3    H_min := 1
4    for τ_i in τ loop
5      H_min := lcm (H_min, l_i)
6    end for
7
8    G := EnumerateLcm (τ, m)
9    d := 1
10   while H_min > min (d * G) loop
11     H_min := FindLcm (τ, d * G, m, H_min)
12     d := d + 1
13   end while
14   return H_min
15 end FHS
```

Listing 2. FHS algorithm

Property 1 *Each $h \in \mathbb{H}$ can be expressed in terms of the last common multiple of g and a set of periods $(t_{m+1}, \ldots, t_n)$ from tasks not in the initial set of m tasks.*

$$\forall h \in \mathbb{H} \ \rightarrow \ \exists g \in \mathbb{G} \ / \ h = lcm(g, t_{m+1}, \ldots, t_n)$$

From this it holds that h is a multiple of g.

Definition 4 *Let $\mathbb{D}$ be the set of values by which some $g \in \mathbb{G}$ is multiple of some $h \in \mathbb{H}$.*

$$\mathbb{D} = \{d \in \mathbb{N} \mid h = dg\}$$

Property 2 *Since each h is a valid hyperperiod, it is possible to find at least one period in the range defined for each task $t_i \in [l_i, u_i]$, which is a divisor of dg.*

$$\forall \tau_i \in \tau \ \rightarrow \ \exists t_i \ \in \ [l_i, u_i] \ / \ dg \ mod \ t_i = 0$$

The algorithm works by increasingly enumerating values of $d \in \mathbb{N}$ and using property 2 to check that they belong to $\mathbb{D}$ by using the `FindLcm` (Listing 3). Additionally, this function returns the minimum value of dg that happens to be a valid hyperperiod h. The algorithm iteratively converges to the minimum hyperperiod H_{min} by storing the value returned by `FindLcm()` for each d. The algorithm stops when the current minimum found is lower that the minimum value for $d * \mathbb{G}$.

Although in each iteration the minimum gd value is calculated, as shown in Figure 1 it may happen that this is not the minimum hyperperiod H_{min}. Let's suppose that there exist g_0 and g_1 that belong to $\mathbb{G}$, and it holds that:

$$g_0 < g_1$$

Let's suppose that $dg_1 \in \mathbb{D}$, that is, dg_1 is a valid hyperperiod. To be the minimum hyperperiod, it must hold that:

$$(d+1)g_0 \geq dg_1,$$

and this may not be true.

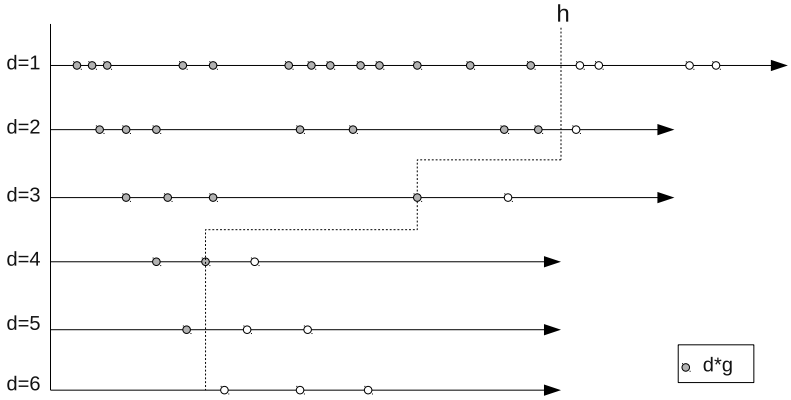

Figure 1. How the algorithm works

```
1  function IsLcm (g, τ_i)
2  begin
3    for t in l_i..u_i
4      if g mod t = 0 then
5        return true
6    end for
7    return false
8  end IsLcm
9
10 function FindLcm (τ, G, m, H_min)
11 begin
12   for g in G loop
13     if g ≥ H_min then
14       return H_min
15     end if
16
17     for i in (m + 1)..n loop
18       found := IsLcm (g, τ_i)
19       if not found then
20         break
21     end for
22     if found then
23       return g
24     end if
25   end for
26   return H_min
27 end FindLcm
```

Listing 3. FindLcm algorithm

4 Conclusions

We have presented an algorithm that allows to calculate the minimum value for the hyperperiod, given that periods of tasks are not given as a values but as ranges of valid values. It has been shown that the algorithms uses a search heuristic that nevertheless is able to found the absolute minimum value of such hyperperiod in most of the cases.

We think that in certain applications the hyperperiod reduction has important benefits and our aim is to continue exploring the presented interpretation of periods as ranges of valid values, in order to continue improving the presented algorithm as well as possibly developing new ones.

References

[1] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

[2] R. Crandall and C. B. Pomerance. *Prime numbers: a computational perspective*. Springer, 2005.

[3] O. Kermia, L. Cucu, and Y. Sorel. Non-preemptive multiprocessor static scheduling for systems with precedence and strict periodicity constraints. In *Proceedings of the 10th International Workshop On Project Management and Scheduling*, 2006.

[4] J. Leung and R. Merrill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 18:115–118, 1980.

[5] C. Liu and J.W.Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 23:46–68, 1973.

[6] C. Macq and J. Goossens. Limitation of the hyper-period in real-time periodic task set generation. In *Proceedings of the 9th international conference on real-time systems*, pages 133–148, March 2001. ISBN 2-87717-078-0.

[7] I. Ripoll, A. Crespo, and A. Mok. Improvement in feasibility testing for real-time tasks. *Journal of Real-Time Systems*, 11:19–40, 1996.

[8] C.-S. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha. Scheduling real-time dwells using tasks with synthetic periods. pages 210 – 219, dec. 2003.