Ripoll Ripoll, JI.; Ballester-Ripoll, R. (2013). Period selection for minimal hyperperiod in periodic task systems. IEEE Transactions on Computers. 62(9):1813-1822. doi:10.1109/TC.2012.243.

Period Selection for Minimal Hyper-period in Real-Time Systems

Ismael Ripoll, Rafael Ballester

Abstract—Task period and deadline selection are often used to adjust the workload to the available computational resources. This paper presents a method for adjusting the periods of the tasks which obtains the minimal hyper-period. Task periods are modelled as a range of possible periods. The selected periods are not constrained to be natural numbers, but can also be rational numbers. Without this limitation, the resulting hyper-period is much smaller than that of previous works. Hyper-period for natural periods grows exponentially; with rational periods the worst case is quadratic with the largest period.
Our finding has practical applications in several fields of real-time scheduling: lowering complexity in table driven schedulers, reducing search space in model checking analysis, generating synthetic workload for statistical analysis of real-time scheduling algorithms, etc.

Index Terms—Real-time systems, Scheduling, Plan generation, Periodic tasks, Hyper-period.

1 Introduction

The periodic task model is the base of real-time scheduling theory. It provides a natural and simple way to describe the behaviour of many physical systems that can also be validated analytically [1]. The basic model consists of two parameters: e_i and p_i ; the worst case execution time (WCET) and the period, respectively.

The hyper-period is defined as the least common multiple (lcm) of the periods of all the periodic tasks: $\mathcal{P} = lcm(p_1 \cdots p_n)$, where n is the number of tasks. Depending on the algorithm used for scheduling the periodic tasks and the parameters of the tasks, the value of the hyper-period may be critical for the analysis or even for the realisation of the system.

Leung et al. [2] showed that feasibility analysis of periodic task systems can be conducted in time $\mathcal{O}(\mathcal{P} \log n)$. Although the schedulability analysis has been later greatly improved, and does not rely on the length of the hyper-period [3], there are still many situations where hyper-period length is a critical issue.

Dynamic priority schedulers perform better than fixed priority ones. EDF (Earliest Deadline First) and LLF (Least Laxity First) are optimal for periodic task sets with arbitrary deadlines. Optimal in the sense that if the task set can not be successfully scheduled by EDF or LLF, then there is no other scheduling algorithm that is able to do it. Although there are fast and efficient schedulability analysis algorithms for EDF [4], [5], unfortunately, the complexity of the analysis is a co-NP-complete in the strong sense [4]. When the task model is enriched with additional requirements and constraints (precedence constraints, non-preemptable tasks, multi-processor, communications, distributed systems, etc.), analytic analysis becomes more and more complex. Eventually, exhaustive state space search over one or more hyper-periods is the only known solution.

Time/table driven policies, which are considered by the research community as too rigid and not appropriate for the current applications demands (complex systems, adaptable, higher processor utilisation, QoS capabilities, etc.) have received little attention. However, table driven solutions are still a valid (or even the only) solution for some complex systems. Ridouard et al. [6] showed that it is not possible to successfully schedule (neither with fixed nor dynamic priorities) tasks allowed to self-suspend, whereas off-line scheduler can be easily realised.

In table-driven scheduling, a table with the actions to be executed is constructed off-line. At run time, the table is cyclically executed. The length of the table depends on the number of actions (tasks or functions) to be executed along with the hyper-period. The shorter the hyper-period, the smaller the table will be. Table driven scheduling is an appealing solution for small systems with scarce computing capabilities [7], but also on complex systems where inter-connected resources of different categories have to be coordinately scheduled. Compared to priority scheduling, table driven schedulers offer: i) simple and fast on-line scheduling algorithms, ii) multiple devices (processor, buses, etc.) can be managed synchronously, iii) easier certification process for highly critical systems. Their main drawback is related to the complexity of building and maintaining the table. A small hyper-period alleviates this problem.

In some cases, simulation is the most convenient method to show the performance of a new algorithm. Statistical comparison of scheduling algorithms must consider an interval proportional to $\mathcal{P}$.

Often it is assumed that the timing constraints of

- *I. Ripoll is with the Institut Tecnològic d'Informàtica, Universitat Politècnica de València, Camí de Vera s/n, 46022 Valàlencia, Spain. E-mail: iripoll@disca.upv.es*
- *R. Ballester is with the CFIS (Centre de Formació Interdisciplinària Superior), Universitat Politècnica de Catalunya.*

asdasd

a real-time task are precisely known at the scheduling analysis phase. This information is obtained from the feedback control algorithms or the operational application parameters, and used as input data for the scheduling analysis. In practical applications, however, this hypothesis is unrealistic [8]. The emerging control-scheduling codesign methodology is trying to overcome this limitation. The goal is to combine the design of the feedback control algorithm with the implementation and scheduling limitations to produce a global optimal solution. Årzén et al. [9] stated the problem as:

> *Given a set of processes to be controlled and a computer with limited computational resources, design a set of controllers and schedule them as real-time tasks such that the overall control performance is optimized.*

The periods of the tasks, and so the hyper-period, are considered as tunable parameters that can be adjusted in an integrated system codesign.

This work presents a novel algorithm for finding the minimal hyper-period of a periodic task set, where periods are specifies as ranges. The algorithm is optimal in the sense that it finds the periods from the given set of ranges, which have the minimal hyper-period. A common assumption is that task parameters are natural numbers. This restriction, which have little or no impact at all in other aspects of scheduling, has a large impact on the value of the hyper-period. We have removed this assumption, and allow periods to be rational numbers. The resulting value for rational periods is, in the worst case, quadratic. While the hyper-period when periods must be natural numbers is exponential.

This paper is organised as follows: next section presents an overview of the state-of-the-art in period selection and hyper-period reduction: section 3 revisits the idea of "periodic task" and defines the bases for the elastic task model presented in section 4; the problem is formally stated in section 5; and the solution presented in section 6; the conclusions are summarised in section 8.

2 State of the art

The idea of selecting task periods appears in the literature as a mean to solve different scheduling problems.

In [10] the authors proposed an integrated approach to controller design and task scheduling. Task frequencies are allowed to vary within a certain range (determined by the stability of the control algorithms). For each task a performance index function which depends on the sampling frequency and the control function. an algorithm is proposed to determine the task frequencies such that all tasks are schedulable for both EDF and RMA. In a posterior work, Seto et al. [11] presented an algorithm to select the periods which optimise the system performance when the tasks are scheduled with the rate monotonic scheduler (RMA). The performance of a task is defined as inversely proportional to the period. It is assumed that the shorter the period the better, and there is no lower period limit. The algorithm generates all the set of periods which are RMA schedulable, and then select those sets that have the smallest periods.

The elastic model proposed by Buttazzo et al. [12] considers the period of a task as a flexible spring with a given rigidity coefficient and an upper and lower limits. The period has a nominal value which is allowed to be adjusted within a range. The computation time is assumed to be fixed. The utilisation factor of a task can be adjusted by changing the period, which can be used to adjust the workload to handle overload situations, increase the overall system utility or dynamically accept new tasks. All this controlling actions are done by a central QoS manager.

The problems related to long hyper-periods, although deserved less attention, have also been studied by a number of authors.

Goossens et al [13] addressed the problem of generating random task sets (to be used on algorithm simulations) whose hyper-period are small enough to simulate the whole hyper-period in a reasonable time. They show that the hyper-period may grow exponentially with both, the greatest possible period, and the number of tasks: n. In particular, when the periods are large, the hyper-period is likely to be a huge number, not practical for simulation purposes.

The periods are generated using a table of numbers which is built by selecting an arbitrary number of primes, and for each prime the maximum allowed exponent. This selection of primes and exponents generates a set base divisors. Each period is then generated multiplying a set of randomly selected divisors. Depending on the selected base primes and the maximum exponents, the resulting tasks sets will show different properties: many different periods or periods that are far from the others, etc.

Recently, Jia Xu [14] presented a method for systematically adjusting the period lengths of periodic tasks, such that the adjusted period are as closely harmonically related to each other (therefore, having a small lcm) as possible. The method consists on building a sorted list of reference periods which are multiple of a selected set of prime numbers and user defined maximum exponents, which are used as base divisors. is then used to adjust the original task periods to the closest reference period. It is guaranteed that the new generated period is greater or equal to the 90% of the original one: $\frac{AdjustedPeriod}{OriginalPeriod} \geq \frac{9}{10}$.

Goosens and Jia Xu use a similar strategy, the former for generating random task sets, and the later to adjust the given workload. The core idea is to build the periods as composite numbers of relatively small primes $(2, 3, 5, 7, 11, \ldots)$.

In [15], Borcal et al. presented a periodic task model where the periods are defined as ranges (min, max values) and presented an heuristic algorithm for finding the set of periods that gives the minimal lcm. The resulting adjusted periods are natural numbers.

3 Reasoning about periodic behaviour

The real-time periodic model defines the workload as a set of n tasks $\{T_1, \cdots, T_n\}$; each task is characterised by the vector $T_i = (e_i, p_i, d_i)$, where e_i is the worst case execution time (WCET) , p_i is the period and d_i the relative deadline.

A task T_i defines an infinite sequence of jobs $J_{i,j}$. Sub-index i identifies the associated task and j is the activation number, $J_{i,k}$ is the k^{th} job in T_i. Each job is defined by the vector $J_{i,j} = (e_i, r_{i,j}, r_{i,j} + d_i)$. Where $r_{i,j}$ is the release time. The release time is the instant in time when the job become ready for execution. Figure 1 sketches the parameters of tasks and jobs.

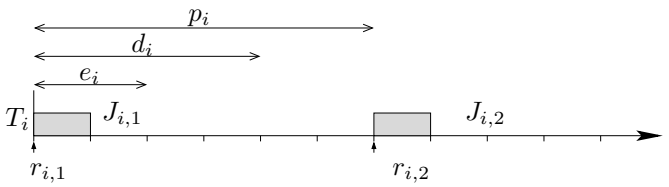


Fig. 1. Classic periodic model

A task set with only one periodic task generates a sequence of jobs that will be executed exactly at the release times, $r_{1,j} = (j-1) \cdot p_1$, and the jobs will be completed e_i time units (t.u. for short) after the release time. Assuming that the processor always uses e_i t.u. and that the scheduler is work-conservative[1].

A periodic task can be more engineering-like defined by saying that it is activity that has to be executed regularly in time. How often the task has to be executed depends on the application. The periodicity can be given as a duration: the time distance between two consecutive executions; or as a frequency: how many times it has to be executed every time unit (second, minute, ...). Also associated to the notion of *periodicity* is the idea of evenly distributed. The events, should be as equally as possible spaced in time. Ideally, events should be p_i t.u. apart from the previous and next one.

While mathematicians work with perfectly defined elements and can assign exact values to the objects they operate with, physical world constraints are not as clearly defined. Engineers are used to work with *tolerance* values. A simple resistor used to have $\pm 10\%$ or $\pm 5\%$ of tolerance with respect to its nominal value; the diameter of a bolt is described as $10 + 0.01 - 0.03$, which gives a range between $[9.97, 10.01]$; etc.

An engineering-like definition of period should contain the value of tolerance jointly with the nominal value. Some examples of periodic work:

- The signal has to be measured every $5sec \pm 10ms$.
- The minimum control frequency is $2Hz$.
- The log file has to be updated three times per hour.

The classic periodic model defines the WCET, period, deadline, etc. as input parameters for the schedulability analysis. Those parameters are referred as: "*temporal requirements*". The task temporal parameters are derived from requirements of the physical world, more precisely from the discrete control algorithms.

A common assumption done by control system designers about the supporting computer system is that it is fast enough to i) measure the sensors, ii) calculate the control response action, and iii) deliver the action to the actuator to execute the three activities all at once, that is, there is no significant delay between reading the sensors and sending the signal to the actuators. This idealised behaviour of a computer is valid when the system dynamics are slow (compared with the processor speed), or when the processor work load is very low. A practical solution to this problem can be found in the automotive industry, where one ECU (electronic control unit) is used for almost every control loop. Another solution came from the control system discipline by developing new multi-frequency, robust control algorithms [16].

But rather than considering scheduling analysis and control system as two separate areas, codesign methodologies consider both as elements of the same problem. The problem is stated by Törngren et al. [17]:

> "Control and cost co-design problem: *Given a set of systems to be controlled and control performance specifications for these, choose an implementation in terms of a distributed computer system including deciding the allocation of control functions, their partitioning into tasks, scheduling and triggering, such that the overall production cost is minimized while guaranteeing the specified control performance.*"

The co-design methodology provides the possibility to operate with the temporal parameters of the tasks in a more flexible way, by adding some of the task parameters to the iterative process of design and implementation.

Summarising, the physical world is commonly analysed, measured and controlled taken into account some degree of inaccuracy or tolerance; the design of a complex real-time system is done in an iterative way, where design choices and even requirements may be revisited or updated several times during the design.

4 The elastic periodic model

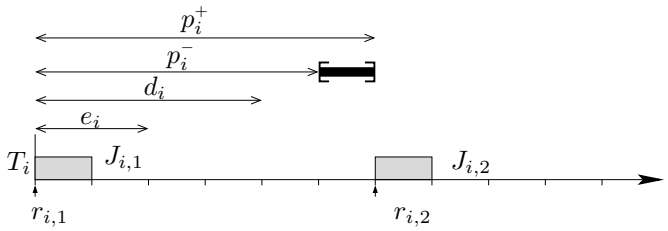


Fig. 2. Elastic model

Our model is a simplification of the Buttazzo's elastic task model [12]. A task is defined as a tuple of four

1. A work-conservative scheduler never idles the processor as long as there are active jobs.

numbers: $T_i = (e_i, [p_i^-, p_i^+], d_i)$. The period is specified as a range of acceptable values, rather than a fixed constant requirement. Once the period is fixed during the analysis phase, the task will be executed at the same constant period as any periodic task of the classic model. In the following, $p_i \in [p_i^-, p_i^+]$ will denote the actual fixed period of the task. The range of periods is only used during the design and analysis phase.

The range of periods will be called *elastic period*. The value of the period that finally will be used to schedule the task is called *fixed period*.

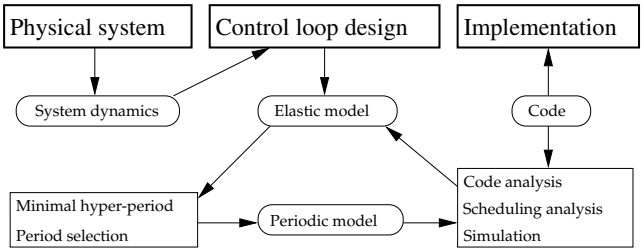


Fig. 3. Design workflow process

Contrarily to Buttazzo's model, the periods of the tasks are not allowed to change on-line. The scheduler, no matter on-line or off-line, will work with periodic tasks with fixed periods.

We will assume that $d_i \leq p_i^-$.

5 Problem Statement

> Given a task set with elastic periods, find the set of periods $p_i \in [p_i^-, p_i^+]$ such that the hyper-period, $\mathcal{P}$, is minimal.

If we force $p_i \in \mathbb{N} \forall i$ then $\mathcal{P} = lcm(p_1, \ldots, p_n)$. This problem is at least as hard as the factorisation problem. Suppose a task set with two tasks whose elastic periods are: $p_1 \in [2\,, \lfloor\sqrt{N}\rfloor]$ and $p_2 \in [N, N]$. If the resulting minimal lcm is N, then N is a composite number, and p_1 is a proper divisor of N. If the minimal lcm is a multiple of N then N is prime. The remaining set of factors can be calculated iteratively with $N' = N/p_1$. An efficient approach to this problem was presented in [15]. It consists of an heuristic that first computes all the lcm's of a subset of the tasks (those with smaller ranges), and then performs an exhaustive search of this set of candidates on the remaining set of tasks.

If we allow the fixed periods p_i to belong to $\mathbb{Q}$ instead of $\mathbb{N}$, the problem can be solved faster and the resulting hyper-period is smaller.

Typically the hyper-period is defined as the lcm of the periods of the tasks, which is a constructive definition. Unluckily, this definition is only valid when the periods are natural numbers. A new definition for hyper-period is needed:

Definition 1: The hyper-period is the smallest positive number that is an integer multiple of every task.

A property of the hyper-period is that each periodic task is activated an integer number of times along the hyper-period: $\mathcal{P}/p_i \in \mathbb{N}$

6 Minimal Hyper-Period for Rational Periods

If the periods are not necessarily integers but are only required to be rational numbers, then the problem is no longer related to the lcm of a set of numbers. And can be reformulated as:

> Given a task set with elastic periods, find the smallest number $\mathcal{P}$ such that exists a set of n integer numbers $\{t_1, \cdots, t_n\}$ such that $\mathcal{P}/t_i \in [p_i^-, p_i^+] \forall i$.

The idea is to define the task period as a fraction of the system hyper-period, rather than as an absolute time duration. For example, if we have a task whose elastic period is $[30, 35]$ and the obtained hyper-period is 100, then the fixed period is $100/3 = 33.\widehat{3}$ t.u., which is equivalent to execute the task 3 times along the hyper-period.

Since the period is a rational number, it is possible to generate a precise sequence of activations that do not cause accumulated drift [2]. For instance, the previous task ($p_i = 100/3$), can be activated at the following times: 0, 33, 67, 100, 133, 167, 200, etc.

Before describing the MinHyperPeriod algorithm, four observations of the properties of the elastic model are formulated:

Observation 1: Each elastic range $[p_i^-, p_i^+]$ defines an infinite list of intervals of valid hyper-periods: $\{[p_i^-, p_i^+], [2p_i^-, 2p_i^+], [3p_i^-, 3p_i^+], \ldots\}$
Let $I_i = \bigcup_{k=1}^{\infty} [kp_i^-, kp_i^+]$ be the set of all valid hyper-period intervals for task T_i.

Any value that falls inside a black interval of figure 4(a) is a valid hyper-period for a task with elastic period $[10, 12]$.

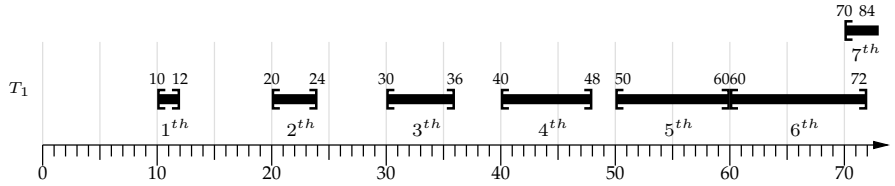
If the resulting hyper-period is a number within the interval $[2p_i^-, 2p_i^+]$, then the task T_i will have two activations in hyper-period; if it is within the interval $[3p_i^-, 3p_i^+]$ then the task will have three activations, and so on.

Observation 2: The intervals generated for each elastic period become wider and wider until they overlap, that is, an interval intersects with the next one.

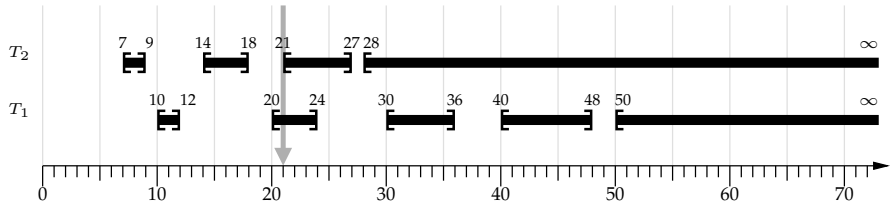
In the figure 4(a), the 5^{th} interval ends at time 60 and the 6^{th} interval starts at 60. The 6^{th} and 7^{th} intervals overlap from 70. Posterior intervals will overlap more and more. Two consecutive intervals of the same task will overlap when $(k+1)p_i^- \geq kp_i^+$. Therefore, there will be at most $\left\lceil \frac{p_i^-}{p_i^+ - p_i^-} \right\rceil$ non-overlapping intervals for each elastic period [3]. The set of valid intervals of the figure 4(a) example are: $\{[10, 12], [20, 24], [30, 36], [40, 48], [50, \infty)\}$.

2. Evenly distribution of the activations along the hyper-period can be done using two methods: 1) floating point arithmetic and 2) the well-known Bresenham algorithm [18], which is much faster (only simple integer operations) and can be used to compute the activation times at run-time.

3. We will assume that $p_i^+ > p_i^-$. This restriction is removed in section 6.3.



(a) Valid intervals (elastic period $[10, 12]$), which overlaps at time 70



(b) Intervals of two tasks: $[10, 12]$ and $[7, 9]$. The minimum hyper-period is 21

Fig. 4. Intervals of valid hyper-periods

Observation 3: The hyper-period must fall inside one interval of each task. Therefore, the hyper-period must belong to the intersection of the task intervals: $\mathcal{P} \in \bigcap_{i=1}^{n} I_i$.

Observation 4: The minimal hyper-period occurs at the first instant of time in which all task intervals intersect: $\mathcal{P} = \min\{\bigcap_{i=1}^{n} I_i\}$.

It is important to differentiate overlapping of intervals of the same task and intersection of intervals of different tasks. When the intervals of a given task overlap, it means that any time after that time will be a valid hyper-period for this task. Therefore, all the following intervals of this task can be merged together into a single interval of the form $[x, \infty)$.

Since the set of intervals of a task define the set of values that are valid hyper-periods for the task, the intersection of intervals of two tasks define the set of valid hyper-periods for that pair of tasks.

In the following, the term *overlap* will refer to the intersection of the intervals of a given task, and term *intersect* will refer to the intersection of intervals of different tasks.

Lemma 1: The value of the hyper-period for rational periods is bounded by:

$$\mathcal{P} \leq \max_{i=1}^{n} \left(p_i^- \left\lceil \frac{p_i^-}{p_i^+ - p_i^-} \right\rceil \right) \simeq \max_{i=1}^{n} \left(\frac{(p_i^-)^2}{p_i^+ - p_i^-} \right)$$

Proof: From observation 2, we know that any period range stars overlapping at time $p_i^- \left\lceil \frac{p_i^-}{p_i^+ - p_i^-} \right\rceil$. Any value after this is a valid hyper-period to task T_i, and any time value after all tasks overlaps is a valid hyper-period for the task set (observation 3). Therefore the task that overlaps latest determines the worst case hyper-period. □

Lemma 2: The minimal hyper-period for rational periods is less than or equal to that obtained considering natural periods only.

Proof: Let $\mathcal{P}$ be the minimal hyper-period obtained when we are looking for natural periods. That is, there exists a set of n integers $\{t_1, \cdots, t_n\}$ such that $\mathcal{P}/t_i = p_i \, \forall i$, where p_i is some natural number belonging to $[p_i^-, p_i^+]$. Thus, $\mathcal{P} = t_i \cdot p_i \, \forall i \Rightarrow \mathcal{P} \in [t_i p_i^-, t_i p_i^+] \, \forall i$; by definition of (I_i) $\mathcal{P} \in I_i \, \forall i \Rightarrow \mathcal{P} \in \bigcap_{i=1}^{n} I_i$; therefore (observation 3) $\mathcal{P}$ is also a valid candidate if we ask for rational periods. Therefore, the minimal hyper-period when periods can be rational numbers will be no larger than $\mathcal{P}$. □

It is interesting to note that a small modification of the model (compatible with the standard en engineering methods) and reformulation of the problem causes a significant change in the obtained solutions. On one hand, the hyper-period of non-elastic task sets grows exponentially with the value of the largest period [13]: $\mathcal{P} \simeq e^{\max(p_i)}$. The probability that the hyper-period of a task set is close to the this limit grows also exponentially with the number of tasks. The larger the task set, the higher the probability to have a high hyper-period. On the other hand, the worst case hyper-period for rational periods is quadratic on $\max(p_i^-)$ with a coefficient smaller than 1.

6.1 MinHyperPeriod algorithm

First, a simplified version of the algorithm is presented and explained (limited to elastic periods), which is improved in subsection6.2. The algorithm is later extended to include both elastic and non-elastic periods. Finally, once the hyper-period is found, subsection 6.4 shows how to calculate the values of the fixed periods.

1) Generate a list of tuples $(time, isStart)$, where $time$ is the value of the lower or upper endpoint time of an interval, and $isStart$ is a Boolean value that identifies whether the time is the lower endpoint (true) or the upper endpoint (false). The intervals of each task must not overlap.
2) Sort the tuples of the list in ascending $time$ order. In the case of the tuples with the same $time$, place the tuples with $isStart = true$ first.

3) Let $counter$ be a variable which counts the number of tasks that have some interval at time t. Initially $counter$ is set to zero.
4) Iterate over the sorted list of tuples. Increment $counter$ every lower endpoint tuple ($isStart = true$) and decrement $counter$ every upper endpoint tuple.
5) The algorithm finishes as soon as $counter = n$. The value of the hyper-period is the value of $time$ of the last visited tuple.

Since the tuples are time ordered, every time a lower endpoint is found we know that at that time we are entering into an interval. It does not matter which is the associated task for that interval, what is needed is just to account that any time from now on (until the next upper endpoint) is a hyper-period candidate for one more task. Likewise, the candidate times lose one task whenever a lower endpoint is encountered.

When the $counter$ reaches the value n, there are n intersecting intervals which belong to n different tasks. Since the tuples are time sorted, the first time that $counter = n$ will be the minimum valid hyper-period.

TABLE 1
Sorted list of tuples and the value of the "counter" for the example on figure 4(b)

time	7	9	10	12	14	18	20	**21**	24	27	···
isStart	T	F	T	F	T	F	T	**T**	F	T	···
counter	1	0	1	0	1	0	1	**2**	-	-	···

The minimal hyper-period is 21 in this table.

6.2 Implementation improvement

Considering that the endpoints are sorted by time and that the intervals of each task do not overlap, it is not necessary to compute the endpoints of all the intervals but only the most recent interval for each task. Therefore, the algorithm only has to keep up a sorted collection of intervals, one for each task. Since each interval is defined by two points, this implies using a list of $2n$ elements.

Every time a task endpoint is removed from the list, the corresponding endpoint of this task's next interval is inserted. For lower endpoint tuples, the next time will be $time := (time + p_i^-)$; and for upper endpoint tuples, the new value of time will be $time := (time + p_i^+)$.

The list of end points can be efficiently implemented with a heap data structure, where insertion and extraction are both $O(log_2(n))$. This algorithm is implemented in the code presented in the appendix.

6.3 Mixed elastic and classic periods

The MinHyperPeriod algorithm is not efficient for non-elastic periods, that is, when the elastic interval has just one value ($p_i^- = p_i^+$). In this case, there is an infinite number of non-overlapping intervals, and the algorithm would degenerate into an exhaustive search of all the period multiples.

Obviously, the hyper-period of the non-elastic periods is the lcm of those periods. Let us denote this value as $\mathcal{P}^0$. Which can be computed as $lcm(a,b) = \frac{a \cdot b}{gcd(a,b)}$. And the gcd (greatest common divisor) can efficiently[4] be computed iteratively using the Euclidean algorithm (mod is the remaining of the division):

$$\begin{aligned} gcd(a,0) &= a \\ gcd(a,b) &= gcd(b, a \bmod b) \end{aligned}$$

The hyper-period of a mixed task set must be a multiple of $\mathcal{P}^0$.

The algorithm for combined task works as follows:

1) Compute the lcm of the non-elastic periods.
2) Run the MinHyperPeriod algorithm with the elastic periods, but the termination of the algorithm will occur when both:
   a) the $counter$ is equal to the number of elastic tasks (initial condition), and
   b) a multiple of $\mathcal{P}^0$ falls inside the current intersecting interval. This occurs if and only if $\lceil \frac{current\_time}{\mathcal{P}^0} \rceil \leq \lfloor \frac{next\_time}{\mathcal{P}^0} \rfloor$. $current\_time$ is the $time$ of the current visited tuple, while $next\_time$ is the time value of the next tuple in the list of endpoint tuples.

The pseudo-code of the appendix also handles mixed task sets.

6.4 Period selection

Once the hyper-period has been obtained, the value of the fixed periods should be calculated. The algorithm guarantees that:

$$\forall i,\ \exists k_i \in \mathbb{N} \ : \ \frac{\mathcal{P}}{k_i} \in [p_i^-, p_i^+]$$

There exists at least one natural number, k_i, the evenly divides $\mathcal{P}$. But k_i may be not unique. There will be multiple values of k_i if $\mathcal{P} > p_i^- \left\lceil \frac{p_i^-}{p_i^+ - p_i^-} \right\rceil$. The values of k_i will be in the range:

$$\lfloor \mathcal{P}/p_i^- \rfloor \leq k_i \leq \lceil \mathcal{P}/p_i^+ \rceil$$

Consider, for example, the set of three tasks of table 2 (although the values of the WCET and deadline are not used to compute neither the value of hyper-period nor the fixed periods, they are specified for completeness).

TABLE 2
Example with multiple period solutions for ($\mathcal{P} = 38$)

Task =	$(c_i,[p_i^-, p_i^+],d_i)$	$\lfloor \frac{\mathcal{P}}{p_i^-} \rfloor$	$\lceil \frac{\mathcal{P}}{p_i^+} \rceil$	Valid periods	
$T_1 =$	$(1,\ [19,20],\ 4)$	2	2	38/2	T_2
$T_2 =$	$(1,\ [12,14],\ 6)$	3	3	38/3	T_2
$T_3 =$	$(1,\ [5,9],\ 15)$	5	7	38/5	T_3^1
				38/6	T_3^2
				38/7	T_3^3

4. Considering the values of the periods in real cases.

As can be seen in figure 5(a), task T_3 intervals overlap at time 10 and the tree tasks intersect at time 38, the minimal hyper-period.

Figure 5(b) sketches the activation pattern of the three tasks. Tasks T_1 and T_2 have only one period solution: $k_1 = 2$ and $k_2 = 3$. But the third task has three solutions: $k_3^1 = 5$, $k_3^1 = 6$ and $k_3^1 = 7$, which correspond with the periods: $p_3^1 = 7.6$, $p_3^1 = 6.3$ and $p_3^1 = 5.4$. It is possible to observe that the hyper-period property is satisfied for all the periods. That is, all the tasks have an activation at time 38.

The final selection of the fixed period for the third task shall be done by the system designed taking into account other aspects not directly related to the processor scheduling as: utility considerations, energy consumption, communication requirements, etc.

As can be seen in this simple example, the use of rational periods has a minimal impact on the "regularity" of the recurrence of the periodic tasks.

6.5 Algorithm complexity

The temporal complexity of the algorithm depends on the number of intervals. Let $I = \sum_{i=1}^{n} |I_i|$ be the total number of non-overlapping intervals of the task set. The first version of the algorithm creates two tuples for each interval; the tuples are sorted; and the list is traversed until $counter = n$. Therefore, the complexity is determined by the cost of sorting I tuples: $O(I \log_2 I)$. From observation 2:

$$I = \sum_{i=1}^{n} \left\lceil \frac{p_i^-}{p_i^+ - p_i^-} \right\rceil$$

The asymptotic complexity of the improved version is: $O(I \log_2 n)$. A tighter cost can be obtained by taking into account that the improved algorithm finishes when the hyper-period is found, and that the loop is iterated twice for every interval (once for the lower endpoint and another for the upper one). Since there are less intervals than activations (due to the fusion of the overlapping intervals after $\left\lceil \frac{p_i^-}{p_i^+ - p_i^-} \right\rceil$), the effective cost is at most the cost of scheduling the task set using fixed priorities during the first hyper-period.

Another way to evaluate the complexity of the improved version is by comparing the operation of the algorithm with a real-time scheduler. The lower and upper endpoint of the intervals match the scheduling points for a classic task set with twice the number of tasks, one for the minimum period and another for the maximum period: $\{T'_1, \cdots, T'_n, T''_1{}_+, \cdots, T''_n\}$; where $T'_i = (e_i, p_i^-, d_i)$ and $T''_i = (e_i, p_i^+, d_i)$

On a mixed task set, the asymptotic complexity is reduced because there are less intervals to be analysed. Interval intersection is only searched for elastic tasks. On the other hand, the mean time may be slightly longer. Non-elastic tasks forces a new condition on the tuple search, which is likely to increase the number of visited tuples before the algorithm terminates.

The spatial complexity of the algorithm, as described in section 6, is linear with the number of tuples, I for the first version, and $2n$ for the improved version.

Both the temporal and spatial cost of the improved version is for practical purposes negligible.

7 Experimental results

First we compare the value of the hyper-period and task periods for the example proposed by [14]. The example contains four tasks ("Flexible P." column); the result of the most accurate solution presented by Xu (Table 7 of his paper) is listed in the column labelled as "Xu". Columns "Natural" and "Rational" show the results of the algorithms of [15] and the current work respectively.

The nominal periods of the Xu's example are the upper endpoints of the respective task intervals (364, 667, 727 and 100000). And the algorithm is allowed to reduce the period up to a defined factor, which in the example is 10%. The nominal period minus the given percentage defines the ranges flexible periodic workload.

TABLE 3
Xu's example solved with Natural and Rational periods, range width 10%

Task	Period	Periods		
		Xu	Natural	Rational
CD-Audio	$[93000, 100000]$	360	93010	93000/1
ISDN	$[677, 727]$	660	710	93000/128
Voice	$[621, 667]$	720	655	93000/140
Keyboard	$[339, 364]$	92400	355	93000/256
Hyper-per.	$> 2^2 10^{13}$	277200	93010	93000

The hyper-period of both the natural and rational periods solution is three times smaller than that of Xu. Also, the fixed periods are closer to the nominal period.

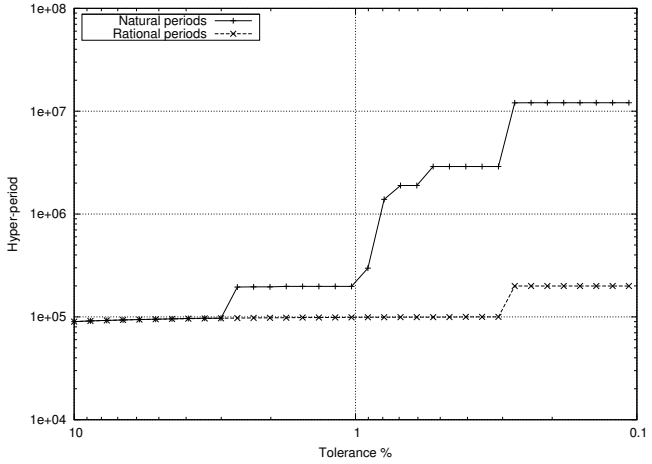
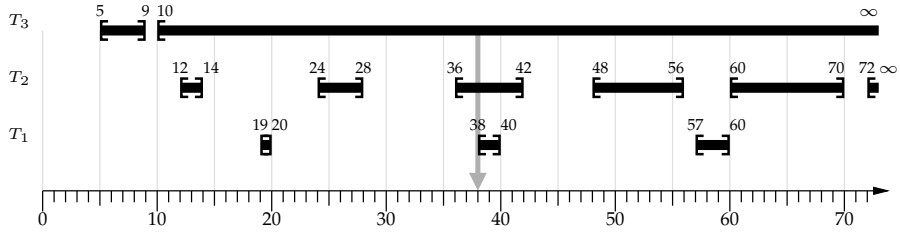
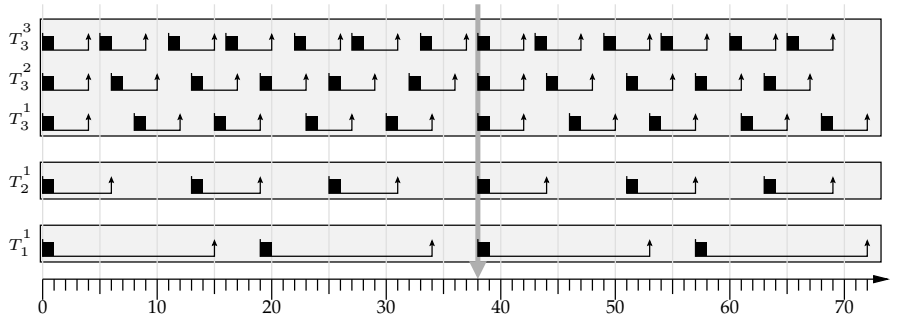


Fig. 6. Impact of the period range width on natural and rational period solutions, on the Xu's example (table 3)

When the ranges are narrow, the solution for natural periods is closer to the lcm of the periods. The solution for rational periods, is exponentially smaller. Using the



(a) Hyper-period computation of task set of table 2



(b) Activation pattern of the obtained fixed periods

Fig. 5. Minimal hyper-period with multiple period solutions

```
1 def GenerateTaskSet(n, Tolerance):
2     MIN=100;
3     MAX=5000;
4     for i in 0...n :
5         p_i^+ = random.uniform(MIN,MAX);
6         p_i^- = p_i^+ - p_i^+ . Tolerance/100;
```

Listing 1. Random workload generation

same workload, figure 6 shows how the minimal hyper-period is affected by the range of the flexible periods. The values for period range equal to 10% correspond to the values on table 3.

The remaining experiments has been designed using randomly generated workload, using the code of listing 1.

Figure 7 shows the value of the minimal hyper-period as a function of the range width, for natural and rational periods. As expected, tighter period ranges result in larger hyper-periods. But while the hyper-period with natural periods grows exponentially, with rational periods the hyper-period grows polynomially.

Figure 8 compares the hyper-period for natural and rational periods when increasing the number of tasks. The lcm of a set of integers shows a strong dispersion depending how many and how large are the divisors or each number; when using elastic periods the dispersion is reduced, but it is still significant as shown by the wide error bars of the figure 8 (The error bars represent the smaller and larger value of the 1000 simulations). When periods can be rational numbers both, the dispersion and the value of the hyper-period is much smaller.

The resulting values with rational periods are less

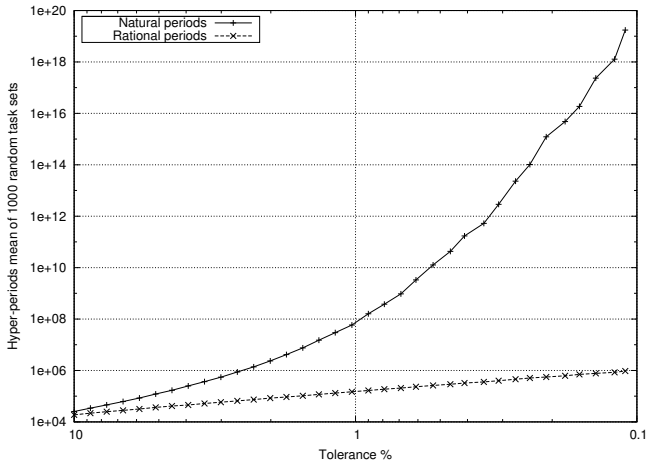


Fig. 7. Natural vs Rational period solution comparison, constant number of tasks ($n = 10$)

affected by the number of tasks and the range of the periods, as shown in figure 9.

As expected from the analytical upper bounds, the use of rational periods significantly reduces the value of the hyper-period.

8 Conclusion

The hyper-period is an important property with implications in a wide variety of real-time scheduling issues. For example, a relatively small hyper-period is beneficial to table driven schedulers because reduces table size, as well as its building and maintaining complexity. Scheduling analysis of priority driven algorithms (RM and EDF) of complex systems may require to analyse as

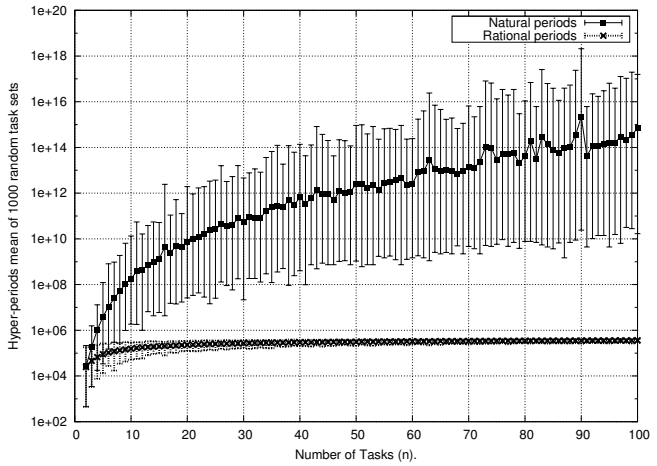


Fig. 8. Natural vs Rational period solution comparison, constant tolerance: ($Tolerance = 1\%$)

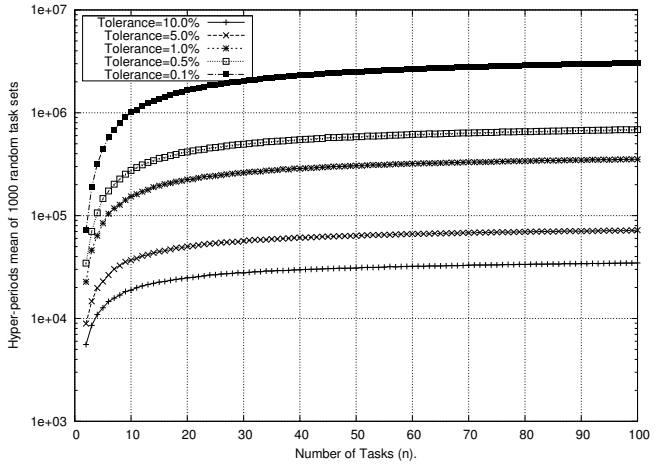


Fig. 9. Hyper-period of rational periods

much events as the number of jobs in the one or two hyper-periods.

Hyper-period reduction is achieved by using a flexible period definition. Each task period is defined as a range of min/max period values. In previous works, the final selected periods are limited to be natural numbers. In this work, the resulting periods are allowed to be fractional numbers, which reduces the cost of the algorithm. And also, the resulting hyper-period is much smaller in the worst case.

There are several open issues to explore. Study the behaviour (worst case, average case and analytic model) of the hyper-period with rational periods in relation with period range widths. Apply this result to multi-resources table driven systems (several processor, buses and devices).

ACKNOWLEDGMENT

This work has been funded by the Spanish Government Research Office, project TIN2008-06766-C03-02 (RT-MODEL).

REFERENCES

[1] L. Sha, T. F. Abdelzaher, K.-E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. C. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-Time Systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

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

[3] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority preemptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

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

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

[6] F. Ridouard, P. Richard, and F. Cottet, "Negative results for scheduling independent hard real-time tasks with self-suspensions," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 47–56. [Online]. Available: http://dl.acm.org/citation.cfm?id=1038265.1038970

[7] B. Miramond and L. Cucu-Grosjean, "Generation of static tables in embedded memory with dense scheduling," in *DASIP*. IEEE, 2010, pp. 105–112.

[8] F. Xia and Y. Sun, "Control-scheduling codesign: a perspective on integrating control and computing," *Dynamics of Continuous, Discrete and Impulsive Systems - Series B: Applications and Algorithms*, vol. 13, no. S1, pp. 1352–1358, 2006.

[9] K.-E. Årzén and A. Cervin, "Control and Embedded Computing: Survey of Research Directions," in *Proceedings of the 16th IFAC World Congress*. Elsevier, Jul. 2005.

[10] D. Seto, J. Lehoczky, L. Sha, and K. Shin, "On task schedulability in real-time control systems," in *IEEE Real-Time Systems Symposium*, 1996.

[11] D. Seto, J. P. Lehoczky, and L. Sha, "Task period selection and schedulability in real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, ser. RTSS '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 188–198.

[12] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *IEEE Real-Time Systems Symposium*, December 1998, pp. 286–295.

[13] 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*, March 2001, pp. 133–148, iSBN 2-87717-078-0.

[14] J. Xu, "A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems," in *Mechatronics and Embedded Systems and Applications (MESA), 2010 IEEE/ASME International Conference on*, july 2010, pp. 288 –294.

[15] V. Brocal, P. Balbastre, R. Ballester, and I. Ripoll, "Task period selection to minimize hyperperiod," in *Proceedings of the 16^{th} IEEE International Conference on Emerging Technologies and Factor Automation*, September 2011.

[16] M. Törngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems*, vol. 14, pp. 219–250, 1998, 10.1023/A:1007964222989. [Online]. Available: http://dx.doi.org/10.1023/A:1007964222989

[17] M. Törngren, D. Henriksson, O. Redell, C. Kirsch, J. El-Khoury, D. Simon, Y. Sorel, H. Zdenek, and K.-E. Årzén, "Co-design of control systems and their real-time implementation — a tool survey," Royal Institute of Technology (KTH), Stockolm, Sweden, Tech. Rep. KTH/MMK/R-06/11-SE, September 2006.

[18] J. E. Bresenham, "Algorithms por computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.