Schedulability Analysis for Non-preemptive Tasks under Strict Periodicity Constraints

Omar Kermia, Yves Sorel

INRIA Rocquencourt,
BP 105 - 78153 Le Chesnay Cedex, France
Phone: +33 1 39 63 52 60 - Fax: +33 1 39 63 51 93
omar.kermia@inria.fr, yves.sorel@inria.fr

Abstract

Real-time systems are often designed using preemptive scheduling to guarantee the execution of high priority tasks. For multiple reasons there is a great interest in exploring non-preemptive scheduling in the case of hard real-time systems where missing deadline leads to catastrophic situations. This paper presents a necessary and sufficient schedulability condition for determining whether a task will satisfy its period and precedences constraints when some tasks have already been scheduled. Tasks we are dealing with are non-preemptive and the periods considered here are strict.

1 Introduction

Hard Real-Time preserves temporal and functional feasibility. Hard real-time system scheduling has been concerned with providing guarantees for temporal feasibility of task execution in all anticipated situations. A scheduling algorithm is defined as a set of rules defining the execution of tasks at system run-time. It is provided thanks to a schedulability or feasibility analysis, which determines, whether a set of tasks with parameters describing their temporal behavior will meet their temporal constraints if executed at run-time according to the rules of the algorithm. The result of such a test is typically a yes or a no answer indicating whether feasibility will be met or not. These schemes and tests demand precise assumptions about task properties, which hold for the entire system lifetime.

In practice, periodic tasks are commonly found in applications such as avionics and process control when accurate control requires continual sampling and processing of data. Such applications based on automatic control and/or signal processing algorithms are usually specified with block-diagrams. They are composed of functions producing and consuming data, and each function can start its execution as soon as data it consumes is available since tasks are not preemptive, and the cost of a task scheduling is a constant included in its worst case execution time (WCET).

Strict period means that if the periodic task A has period T_A then $\forall i \in \mathbb{N}, (s_{A_{i+1}} - s_{A_i}) = T_A$, where A_i and A_{i+1} are the i^{th} and the $(i+1)^{th}$ repetitions of the task A, and s_{A_i} and $s_{A_{i+1}}$ are their start times [1].

Given a real-time system of tasks, the goal is to determine, offline, a monoprocessor static scheduling of the tasks, such that each task completes its execution before a specified deadline that we take here equal to its strict period. Since tasks are deduced from functions producing and consuming data we consider in this case that they are dependent. Finally, due to the hard real-time nature of the considered systems we deal with non-preemptive tasks. Non-preemptive scheduling is important for a variety of reasons [2]:

- In many practical real-time scheduling problems such as I/O scheduling, properties of device hardware and software either make preemption impossible or prohibitively expensive. The preemption cost is either not taken into account or still not really controlled;
- Non-preemptive scheduling algorithms are easier to implement than preemptive algorithms, and can exhibit dramatically lower overhead at run-time;
- The overhead of preemptive algorithms is more difficult to characterize and predict than that of non-preemptive algorithms. Since scheduling overhead is often ignored in scheduling models, an implementation of a non-preemptive scheduler will be closer to the formal model than an implementation of a preemptive scheduler.

For these reasons, designers often use non-preemptive approaches, even though elegant theoretical results do not extend easily to them [3].

As shown in [4] to analyze a system composed of non-preemptive periodic tasks it is enough to study its behavior for a time interval equal to the least common multiple (LCM) of all the task periods, called the hyper-period. Consequently, each task of the initial system will be repeated according to the ratio between its period and the hyper-period. Notice that in general, the value of the hyper-period is not large due to the relatively small number of sensors and actuators which impose their periods to the tasks [5]. Thus, the resulting system we have to deal with will not be significantly larger than the initial one.

According to [6] we also assume that the dependent tasks must be at the same period or at multiple periods in order for the consumer task to be able to receive the data sent by the producer task without some data being lost or duplicated. This restriction does not prevent the presence of tasks with non multiple periods in the same system, nevertheless these tasks must not be dependent [7]. In this case, the data mechanism can be explained as follows, when two tasks are dependent, and do not have the same period, there are two possibilities:

- if the period of the consumer task is equal to n times the period of the producer task then the producer task must be executed n times compared to the consumer task, and the consumer task cannot start its execution until it has received all data from the n executions of the producer task (we have to precise that the produced data differ from one execution of the producer task to another execution therefore data are not duplicated);

- reciprocally, if the period of the producer task is equal to n times the period of the consumer task then the consumer task must be executed n times compared to the producer task.

The rest of the paper is organized as follows: the next section is devoted to the related work. Section 2 gives the model used. In section 3 we introduce the schedulability analysis through several theorems and a corollary. After that, section 4 gives an example for the proposed method. Finally, section 5 presents a conclusion.

2 Related Work

It exists a lot of monoprocessor schedulability analysis for popular algorithms like RM and EDF [8]. Unfortunately, as we do not deal with the same models as in these algorithms, their schedulability conditions become, at best, a necessary conditions. Let's take the example of EDF's schedulability condition which is $\sum_{i=0}^{n} \frac{E(o_i)}{T(o_i)} \leq 1$ for a set of n tasks where $E(o_i)$ and $T(o_i)$ are respectively the execution time and the period of a task o_i [8]. This condition is just a necessary condition for our problem since it is a specific case of the problem for which this condition is used [9].

The problem of knowing whether, a non-preemptive set of periodic tasks under the constraints defined previously is schedulable has been shown NP-Complete in the strong sens by [2].

The main difference between what is done in [2] and the present work consists in the definition of the period. In [2] the authors consider that the k^{th} execution of a periodic task ρ with a period p must begin no earlier than t_k and be completed no later than the deadline of $t_k + p$. Whereas in our model, and due to strict period constraint, ρ must begin exactly at t_k and no lag is allowed. Therefore, the scheduling problem, as it is presented here, does not have any solution in the literature.

3 Real-time system model

We deal with systems of real-time tasks with precedence and strict periodicity constraints. A task o is characterized by a period $T(o)$, a worst case execution time $E(o)$ with $E(o) \leq T(o)$, and a start time $S(o)$. By strict periodicity constraint we mean that if the periodic task o has period $T(o)$ then $\forall i \in \mathbb{N}, S(o_{i+1}) - S(o_i) = T(o)$, where o_i and o_{i+1} are the i^{th} and the $(i+1)^{th}$ repetitions of the task o, and $S(o_i)$ and $S(o_{i+1})$ are their start times.

The precedences between tasks are represented by a directed acyclic graph (DAG) denoted G which is the pair $(\mathbb{V}, \mathbb{E})$. $\mathbb{V}$ is the set of tasks characterized as above, and $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ the set of edges which represents the precedence (dependence) constraints between tasks. Therefore, the directed pair of tasks $(o_i, o_j) \in \mathbb{E}$ means that o_j must be scheduled, only if o_i was already scheduled and we have $S(o_i) + E(o_i) \leq S(o_j)$.

We assume that the periods and WCETs are multiple of a unit of time $U = 1$ which means that they are integers representing for example some cycles of the processor clock. If a task o with execution time $E(o)$ is said to start at time unit t, it starts at the beginning of time unit t and completes at the end of time unit $t + E(o) - 1$. Reciprocally, a time interval $[t_1, t_2]$ denotes a set of consecutive time units, given by $\{t_1, t_1 + 1, ..., t_2\}$.

4 Schedulability Analysis

The schedulability analysis consists in verifying that a task could be scheduled with other tasks already proved schedulable using the same schedulability analysis. A task is schedulable means that it exists one or several time intervals on which this task can be scheduled, i.e. its period and periods of already schedulable tasks are satisfied.

It is known that in general a system of tasks can be scheduled by some algorithms whereas other algorithms fail to do it. Consequently, the schedulability analysis depends on the chosen scheduling algorithm. We chose a scheduling algorithm which satisfies the strict periodicity of tasks in addition to the precedences constraints. This algorithm is based on the following ideas:

- tasks are sorted according to the precedence order. If several tasks are in parallel (there are no precedences between them) the tasks with the smallest period has the highest priority.
- before scheduling a task, we must ensure that this task satisfies all periods of tasks already scheduled. Otherwise the start time of this task will be delayed (see example 1)
- a task is scheduled immediately after the completion of the previous one, idle times are introduced only when it is required.

Example 1 *In figure 1 we applied the previous algorithm on a system of tasks composed of five periodic tasks:* $(a : 1, 4), (b : 1, 6), (c : 1, 8), (d : 1, 12), (e : 1, 12)$ *connected by some precedences (see figure 1-1=). We consider a scheduling on the hyper-period of these tasks which is the* $LCM(4, 6, 8, 12) = 24$.

Figure 1-2 depicts the resulting scheduling using the given algorithm. As we can see, tasks are schedulable, while precedence and strict periodicity constraints are satisfied. As it is indicated by the third item of the previous paragraph, first instances of tasks are scheduled on the first interval which satisfies the period of tasks already scheduled. We observe that the start time of the task e_1 *is delayed by one interval of length* 1 *in order to allow the execution of task* a_2. *After that the execution of other instances of tasks follow their periods.*

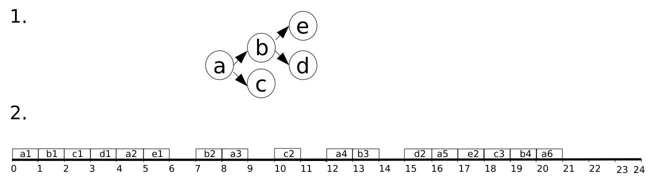

Figure 1. Tasks Graph and The Resulting Scheduling

Starting with two tasks, theorem 1 gives a first schedulability condition. Then, we prove in theorem 2 that condition of theorem 1 is not applicable to n tasks. In order to generalize this condition to n tasks several theorems are necessary. First, theorem 3 gives a schedulability condition for tasks which have, two by two, the same GCD (Greater Common Divisor). Second, for two tasks, theorem 4 gives the number of time intervals where the second task is schedulable, the first one being scheduled. Third, theorem 5 generalizes the previous condition for tasks which have, two by two, the same GCD. Finally, theorem 6 and corollary 1 generalize the previous condition, giving a condition for n tasks. All these conditions hold for tasks with precedence constraints, this is proved in theorem 7.

The following theorem gives a necessary and sufficient condition such that two tasks are schedulable.

Theorem 1 *Two tasks* $(o_1 : E(o_1), T(o_1))$ *and* $(o_2 : E(o_2), T(o_2))$ *are schedulable if and only if*

$$E(o_1) + E(o_2) \leq GCD(T(o_1), T(o_2)) \qquad (1)$$

Proof
Let $g = GCD(T(o_1), T(o_2))$ be the greatest common divisor. We start by proving that (1) is a sufficient condition. Let assume that $S(o_1) = 0$ and $S(o_2) = E(o_1)$. Each instance of the task o_1 is executed in an interval belonging to an intervals set I_1, with I_1=$\{\forall n \in \mathbb{N}, [ng, ng + E(o_1) - 1]\}$, each instance of the task o_2 is executed in an interval belonging to an intervals set I_2, with I_2=$\{\forall n \in \mathbb{N}, [ng + E(o_1), ng+E(o_1)+E(o_2)-1]\}$. If we consider a scheduling on $P = LCM(T(o_1), T(o_2))$ then $n = 1..P/g$. If $g \geq E(o_1) + E(o_2)$ no intervals of I_1 and I_2 overlap, i.e. if we decompose the interval $[0, P]$ into P/g intervals of length g, then, due to their periods, an instance of o_1 and an instance of o_2 will be executed into a same interval of length g. So $g < E(o_1) + E(o_2)$ means that I_1 and I_2 overlap. This proves the sufficiency of (1).

To prove the necessity of (1) we show that if $g < GCD(T(o_1), T(o_2))$, tasks o_1 and o_2 cannot be scheduled. Let assume that $g < GCD(T(o_1), T(o_2))$ and without loss of generality we assume also that $S(o_1) = 0$.

Tasks o_1 and o_2 cannot be scheduled means that it exists two integers x, y for which

$$[xT(o_1), xT(o_1) + E(o_1) - 1] \cap$$
$$[S(o_2) + yT(o_2), S(o_2) + yT(o_2) + E(o_2) - 1] \neq \emptyset$$

which is equivalent to,

$$[xT(o_1) - yT(o_2), xT(o_1) - yT(o_2) + E(o_1) - 1] \cap$$
$$[S(o_2), S(o_2) + E(o_2) - 1] \neq \emptyset$$

According to Bezout's theorem [10], there exist two integers p , q for which $pT(o_1) + qT(o_2) = g$. By choosing $x = np$ and $y = -nq$, $n \in \mathbb{N}$ we have

$$[ng, ng + E(o_1) - 1] \cap [S(o_2), S(o_2) + E(o_2) - 1] \neq \emptyset$$

This is true if free intervals between the intervals $[ng, ng + E(o_1) - 1], n = 0, 1, ..$ are of length $g - E(o_1)$, while the intervals $[S(o_2), S(o_2) + E(o_2) - 1]$ are of length $E(o_2)$. Therefore, the assumption that $g < E(o_1) + E(o_2)$ implies that an integer n necessarily exists for which $[ng, ng+E(o_1)-1]$ and $[S(o_2), S(o_2)+E(o_2)-1]$ overlap. This completes the proof of the theorem 1 $\boxdot$

The question is: can we generalize the theorem 1 for more than two tasks?

Only the condition of the theorem 1 is used in the schedulability analysis (start times are not known yet). But we can easily observe that this condition cannot be useful since it compares only two tasks at the same time. Thus, we need a more general condition which takes into account a set of tasks. the following theorem proves that theorem 1 is generalizable only for tasks for which the greatest common divisors of their periods taken two by two are the same.

The generalization of theorem 1 for all tasks gives only a sufficient condition for scheduling them. This condition is proposed by the following theorem.

Theorem 2 *Tasks of the set* $\{i \in \mathbb{N}, i \leq n, (o_i : E(o_i), T(o_i))\}$ *are schedulable if*

$$\sum_{i=0}^{n} E(o_i) \leq GCD(\forall i, T(o_i)) \qquad (2)$$

Proof
Let $g = GCD(T(o_0), .., T(o_n))$. To prove that 2 is a sufficient condition we proceed as for theorem 1. Let assume that $S(o_0) = 0$ and $S(o_i) = E(o_0) + .. + E(o_{i-1}), \imath = 0..n$. Every task o_i $i = 0..n$ is executed in a subset of the set I_i =$\{\forall l \in \mathbb{N}, [lg + S(o_i), lg + S(o_i) + E(o_i - 1]\}$, As $\sum_{i=0}^{n} E(o_i) \leq GCD(T(o_0), .., T(o_n))$ no intervals of I_1, I_2, .. and I_n overlap, which proves the sufficiency of condition (2) $\boxdot$

To check that condition (2) is not a necessary one it suffices to take a simple example with three tasks, one task with period 2 and the two others with period 4. If we put 1 as the execution time of all tasks then these tasks are schedulable even if the sum of their execution times is 3 and it is greater than the GCD$(4, 4, 2) = 2$

The use of condition (2) to schedule tasks may fail to identify a lot of schedulable cases. Consequently the schedulability analysis would be inefficient.

Instead of looking for a condition dealing with all tasks which is an intractable problem [11] we preferred to proceed differently. The goal is, for each candidate task, to check if it is schedulable with other tasks which have already been proved schedulable. We seek a necessary and sufficient condition to do that.

Theorem 3 *Let* $\{\forall i \in \mathbb{N}, i \leq n, (o_i : E(o_i), T(o_i))\}$ *be a set of* n *tasks which satisfy for each two tasks* $(o_a : E(o_a), T(o_a))$ *and* $(o_b : E(o_b), T(o_b))$, $GCD(T(o_a), T(o_a))$ *is the same and let assume that it is equal to* g*. Tasks of this set are schedulable if and only if*

$$\sum_{i=0}^{n} E(o_i) \leq g \qquad (3)$$

Proof
If we consider a scheduling on $P = LCM(\forall i, T(o_i))$ which is the hyper-period, from the P/g intervals of length g, indeed, there will be necessarily one interval I from the previous intervals which will contain all tasks. From this, if the sum of execution times of these tasks is larger than g then it means that these tasks cannot be scheduled on the interval I whereas the periods impose it. Therefore, the sum of tasks execution times must be less or equal to g, which proves the sufficiency of 3.

We prove the necessity of 3 by showing that, if $g < \sum_{i=0}^{n} E(o_i)$, then tasks of the set $\{\forall i \in \mathbb{N}, i \leq n, o_i\}$ cannot be scheduled. To do that we use a proof by induction.

The base case: for a set with two task $\{o_0, o_1\}$ the necessity was proved in theorem 1.

The inductive step: now we show that if the statement holds for the set $\{o_0, o_1, ..., o_{n-1}\}$ then it also holds for the set $\{o_0, o_1, ..., o_{n-1}, o_n\}$.

Let assume that $g < \sum_{i=0}^{n} E(o_i)$ and we have to prove that integers $x_0, x_1, ..., x_n$ exist for which:

$$\begin{gathered}([S(o_0) + x_0T(o_0), S(o_0) + x_0T(o_0) + E(o_0) - 1] \cup ... \cup \\ [S(o_{n-1}) + x_{n-1}T(o_{n-1}), \\ S(o_{n-1}) + x_{n-1}T(o_{n-1}) + E(o_{n-1}) - 1]) \cap \\ [S(o_n) + x_nT(o_n), S(o_n) + x_nT(o_n) + E(o_n) - 1] \neq \emptyset\end{gathered}$$

it is equivalent to,

$$\begin{gathered}([S(o_0) + x_0T(o_0), S(o_0) + x_0T(o_0) + E(o_0) - 1] \cap \\ [S(o_n) + x_nT(o_n), S(o_n) + x_nT(o_n) + E(o_n) - 1]) \cup ... \cup \\ ([S(o_{n-1}) + x_{n-1}T(o_{n-1}), \\ S(o_{n-1}) + x_{n-1}T(o_{n-1}) + E(o_{n-1}) - 1] \cap \\ [S(o_n) + x_nT(o_n), S(o_n) + x_nT(o_n) + E(o_n) - 1]) \neq \emptyset\end{gathered}$$

or equivalently,

$$([S(o_0) + x_0T(o_0) - x_nT(o_n),$$
$$S(o_0) + x_0T(o_0) - x_nT(o_n) + E(o_0) - 1]$$
$$\cap[S(o_n), S(o_n) + E(o_n) - 1]) \cup ...\cup$$
$$([S(o_{n-1}) + x_{n-1}T(o_{n-1}) - x_nT(o_n),$$
$$S(o_{n-1}) + x_{n-1}T(o_{n-1}) - x_nT(o_n) + E(o_{n-1}) - 1]\cap$$
$$[S(o_n), S(o_n) + E(o_n) - 1]) \neq \emptyset$$

As assumes theorem 3 (for each two tasks $(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$, $GCD(T(o_a), T(o_a))$ is the same and it is g) and As tasks of the set $\{o_0, o_1, ..., o_{n-1}\}$ are schedulable (induction's assumption) each two intervals from the set of intervals $\{[S(o_0) + x_0T(o_0), S(o_0) + x_0T(o_0) + E(o_0) - 1], ..., [S(o_{n-1}) + x_{n-1}T(o_{n-1}), S(o_{n-1}) + x_{n-1}T(o_{n-1}) + E(o_{n-1}) - 1]\}$ do not overlap.

In addition, according to Bezout's theorem, it exists pairs of integers (p_i, q_i) for which $p_iT(o_i) + q_iT(o_n) = g$ $(i = 0, ..., n-1)$, by choosing $x_i = l_ip_i$ and $x_n = -l_iq_i$, $n \in \mathbb{N}$ we have

$$([S(o_0) + l_0g, S(o_0) + l_0g + E(o_0) - 1]\cap$$
$$[S(o_n), S(o_n) + E(o_n) - 1]) \cup ...\cup$$
$$([S(o_{n-1}) + l_{n-1}g, S(o_{n-1}) + l_{n-1}g + E(o_{n-1}) - 1]\cap$$
$$[S(o_n), S(o_n) + E(o_n) - 1]) \neq \emptyset$$

Which can be rewritten in,

$$([S(o_0) + l_0g, S(o_0) + l_0g + E(o_0) - 1] \cup ...\cup$$
$$[S(o_{n-1}) + l_{n-1}g, S(o_{n-1}) + l_{n-1}g + E(o_{n-1}) - 1])$$
$$\cap[S(o_n), S(o_n) + E(o_n) - 1] \neq \emptyset$$

Clearly, this must be the case since free intervals between the intervals $([S(o_0) + l_0g, S(o_0) + l_0g + E(o_0) - 1] \cup ... \cup [S(o_{n-1}) + l_{n-1}g, S(o_{n-1}) + l_{n-1}g + E(o_{n-1}) - 1])$ are of length $(g - \sum_{i=0}^{n-1} E(o_i))$, while the intervals $[S(o_n), S(o_n) + E(o_n) - 1]$ are of length $E(o_n)$. The assumption that $g < \sum_{i=0}^{n} E(o_i)$ implies that integers l_i necessarily exist for which $([S(o_0)+l_0g, S(o_0)+l_0g+E(o_0)-1]\cup...\cup[S(o_{n-1})+l_{n-1}g, S(o_{n-1})+l_{n-1}g+E(o_{n-1})-1])$ and $[S(o_n), S(o_n) + E(o_n) - 1]$ overlap. This completes the proof by induction and at the same time it completes the proof of the theorem $\boxdot$

By theorem 3 we have a schedulability condition for some tasks which satisfy some properties but we do not have a condition which deals with all tasks whatever their periods are.

We choose to group, according to theorem 3, schedulable tasks and to look for a new condition which takes into account the candidate task and a tasks group that has been proved schedulable.

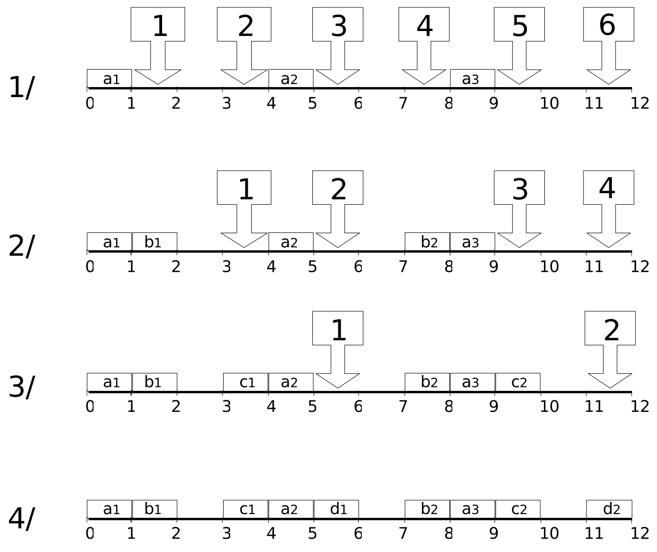

Figure 2. Scheduling Time Intervals

At first, we wanted to know, once a task o_a is scheduled, by taking into account all the time intervals occupied by the instances of this task, how many time intervals can be used to execute another task o_b? For example, if we have a task $(a : 1, 4)$ then the scheduling of its instances yields 6 time intervals on which the task $(b : 1, 6)$ can be scheduled (see figure 2-1). Once that task b is scheduled, there remain 4 time intervals upon which the task $(c : 1, 6)$ can be scheduled (see figure 2-2). After that task c has been scheduled, there remain 2 time intervals upon which the task $(d : 1, 6)$ can be scheduled (see figure 2-3). Finally, after task d has been scheduled there is no other time intervals upon which the task $(e : 1, 6)$ can be scheduled (see figure 2-4). We can observe that tasks b, c and d satisfy one by one the condition (1) with task a.

From this example we know that as soon as a first task o has been scheduled, it remains several intervals of length $E(o)$ where this task could be scheduled or another task which has the same period and WCET as o. The following theorem gives an equation which allows us to compute the number of these intervals.

Theorem 4 *Let $(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$ be two tasks satisfying condition (1) and let*

$g = GCD(T(o_a), T(o_b))$. *If task o_a has been scheduled then the number of available time intervals (equal to $E(o_b)$) in which we can schedule task o_b is given by:*

$$\frac{T(o_b)}{g} \left\lfloor \frac{(g - E(o_a))}{E(o_b)} \right\rfloor$$

Proof
$\dfrac{(g - E(o_a))}{E(o_b)}$ is the number of intervals of length $E(o_b)$ which are able to contain task o_b in one interval of length g and $\left\lfloor \dfrac{T(o_b)}{g} \right\rfloor$ is the number of sub-intervals of length g in one interval of length equal to $T(o_b)$ $\boxdot$

Theorem 4 can be generalized to all tasks that have the same property as tasks of theorem 3.

Theorem 5 *Let (o_{cdt} : $E(o_{cdt}), T(o_{cdt})$) be a candidate task for the schedulability analysis and $\{\forall i \in \mathbb{N}, i \leq n, (o_i : E(o_i), T(o_i))\}$ be a set of n tasks satisfying, for each two tasks $(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$, $GCD(T(o_a), T(o_b))$ is equal to g and no $T(o_i), \forall n$ divides $T(o_{cdt}$. If all tasks o_i were proved schedulable and o_{cdt} satisfies the following condition: $\sum_{i=0}^{n} E(o_i) + E(o_{cdt}) \leq GCD(g, T(o_{cdt}))$ then the number of available time intervals in which we can schedule task o_{cdt} is given by:*

$$\frac{T(o_{cdt})}{GCD(g, T(o_{cdt}))} \left\lfloor \frac{GCD(g, T(o_{cdt})) - \sum_{i=0}^{n} E(o_i)}{E(o_{cdt})} \right\rfloor$$

Proof
As in the previous proof $\left\lfloor \frac{GCD(g,T(o_{cdt})) - \sum_{i=0}^{n} E(o_i)}{E(o_{cdt})} \right\rfloor$ represents the number of intervals of length $E(o_{cdt})$ which are able to contain task o_{cdt} in one interval of length $GCD(g, T(o_{cdt}))$ and $\frac{T(o_{cdt})}{GCD(g,T(o_{cdt}))}$ represents the number of sub-intervals of length $GCD(g, T(o_{cdt}))$ in one interval of length equal to $T(o_{cdt})$ $\boxdot$

Example 2 *Let* $(a : E(a) = 1, T(a) = 8)$, $(b : E(b) = 1, T(b) = 12)$ *and* $(c : E(c) = 1, T(c) = 16)$ *be three tasks already proved schedulable. We want to schedule a task* $(d : E(d) = 1, T(d) = 20)$ *and using theorem 5 compute the number of time intervals where it is possible to schedule it.*

Before starting computing, we check that tasks a,b and c verify conditions. So $GCD(T(a), T(b)) = GCD(T(b), T(c)) = GCD(T(c), T(a)) = 4$, $E(a) + E(b) + E(c) + E(d) = 4 \leq 4$ *and $T(d)$ is not a multiple of a,b and c periods.*

By applying theorem 5 equation we will have : $\frac{20}{4} * \left\lfloor \frac{4-3}{1} \right\rfloor = 5$

Figure 3 shows the five intervals where task d can be scheduled and on intervals where it cannot be scheduled it gives the task which is in conflict with task d.

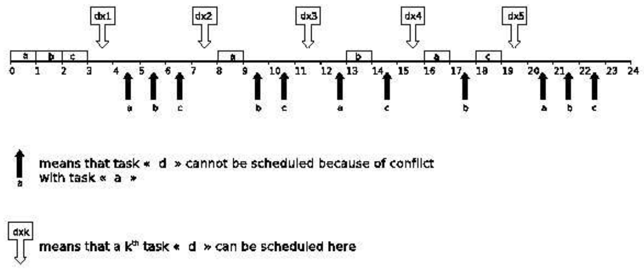
dx1
dx2
dx3
dx4
dx5
a
b
c
a
b
a
c
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
a b c b c a c b a b c
means that task « d » cannot be scheduled because of conflict with task « a »
dxk
means that a k^th task « d » can be scheduled here

Figure 3. Time Intervals

Notation
Because we will divide the set Ω of already proved schedulable tasks in several subsets $\{\Omega_1, \Omega_2, ..., \Omega_m\}$ according to the GCD of each tasks subset, so we denote by $\boldsymbol{BG}$ (basic GCD) the GCD of all tasks of Ω. Let Ω_1 be the set of tasks that satisfy the following property: for each two tasks $(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$ in Ω_1, $GCD(T(o_a), T(o_b))$ is equal to $\boldsymbol{BG}$, $\sum_{o_k \in \Omega_1} E(o_k)$ is denoted by $\boldsymbol{BE}$.

Now, to get the right number of intervals of length $E(o_{cdt})$ and to know if this task is schedulable or not we have to take into account the remainder tasks already proved schedulable but not included in theorem 5. In other words, from the intervals number found using theorem 5 we have to subtract the intervals which lead to conflicts between o_{cdt} and the other tasks already proved schedulable. We mean by conflict whether the scheduling of the candidate task prevents, one or several tasks, to be scheduled following their periods.

Reminder tasks we are talking about are tasks which satisfy: the GCD of any one from them and the candidate task o_{cdt} is different from $\boldsymbol{BG}$.

First, these tasks will be grouped on several subsets, in every subset the following property must be satisfied: for every two tasks $(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$, $GCD(T(o_a), T(o_a))$ is the same. Then, for every subset Ω_j we compute the number $\Gamma(\Omega_j)$ which is the number of intervals of length $E(o_{cdt})$ where o_{cdt} cannot be scheduled because of conflicts with tasks of Ω_j.

The following theorem introduce a way to compute Γ for a given subset.

Theorem 6 *let* Ω_j=$\{2 \leq j \leq m, (o_j : E(o_j), T(o_j))\}$ *be a set of m tasks satisfying, for each two tasks*

$(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$, $GCD(T(o_a), T(o_b))$ is equal to g. If $(o_{cdt} : E(o_{cdt}), T(o_{cdt}))$ is a candidate task for the schedulability analysis then the number of intervals of length $E(o_{cdt})$ where o_{cdt} cannot be scheduled because of conflicts with Ω_j's tasks: let's $Q = (\boldsymbol{BG} - \boldsymbol{BE})$ and $E(\Omega_j) = \sum_{j=0}^{m} E(o_j)$

$$\Gamma(\Omega_j) = \frac{T(o_{cdt})}{GCD(g, T(o_{cdt}))}\left[\left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor \left\lfloor \frac{E(\Omega_j)}{Q} \right\rfloor + \left(\left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor - \left\lfloor \frac{E(\Omega_j) - E(\Omega_j) \bmod Q}{E(o_{cdt})} \right\rfloor\right)\right]$$

Proof
$\frac{T(o_{cdt})}{GCD(g,T(o_{cdt}))}$ is the number of of sub-intervals of length $GCD(g, T(o_{cdt})$ in one interval of length equal to $T(o_{cdt})$. This result is multiplied by $\left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor \left\lfloor \frac{E(\Omega_j)}{Q} \right\rfloor + \left(\left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor - \left\lfloor \frac{E(\Omega_j)-E(\Omega_j) \bmod Q}{E(o_{cdt})} \right\rfloor\right)$
which represents in one interval of length $GCD(g, T(o_{cdt})$ the number of sub-intervals of length $E(o_{cdt})$ where o_{cdt} cannot be scheduled because it prevents one or several tasks from Ω_j to satisfy their periods. This calculation assumes (following the scheduling algorithm described previously) that each task from Ω_j will be scheduled immediately after the completion of the previous scheduled one ⊡

Now, using the result of the two previous theorems, we introduce a condition which allows us to verify the schedulability, according to the proposed algorithm, during the schedulability analysis. The following corollary proposes a way to know if a task o_{cdt} is schedulable.

Corollary 1 *Let $\{\Omega = \forall i \in \mathbb{N}, i \leq n, (o_i : E(o_i), T(o_i))\}$ be a set of tasks already proved schedulable. Let assume that from this set we can set up m subsets according to property of theorem 3 with for each subset $\Omega_j, j \leq m$ a greatest common divisor $g_j, j \leq m$. Let $\boldsymbol{BG}$ be the greatest common divisor of set Ω and the subset Ω_1, $\boldsymbol{BE} = \sum_{o_k \in \Omega_1} E(o_k)$ and $Q = (\boldsymbol{BG} - \boldsymbol{BE})$.*

Let assume that $(o_{cdt} : E(o_{cdt}), T(o_{cdt}))$ is a candidate task for the schedulability analysis then the number of intervals of length $E(o_{cdt})$ where o_{cdt} can be scheduled by the proposed scheduling algorithm is equal to: $A - B$

where:

$$A = \frac{T(o_{cdt})}{\boldsymbol{BG}} \left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor$$

and

$$B = \sum_{j=2}^{m} \Gamma(\Omega_j)$$

$$= \sum_{j=2}^{m} \frac{T(o_{cdt})}{GCD(g_j, T(o_{cdt}))}\left[\left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor \left\lfloor \frac{E(\Omega_j)}{Q} \right\rfloor + \left(\left\lfloor \frac{Q}{E(o_{cdt})} \right\rfloor - \left\lfloor \frac{E(\Omega_j) - E(\Omega_j) \bmod Q}{E(o_{cdt})} \right\rfloor\right)\right]$$

o_{cdt} can be scheduled if and only if
$A - B \leq 0$.

Proof
This corollary is the logical outcome of the two previous theorems. First, among already proved schedulable tasks, we take into account only tasks which satisfy the following conditions: for each two tasks $(o_a : E(o_a), T(o_a))$ and $(o_b : E(o_b), T(o_b))$, $GCD(T(o_a), T(o_b)) = \boldsymbol{BG}$. These tasks constitute the basis of the scheduling. The schedulability analysis of all the other tasks consists in, first, checking the condition of theorem 3 with these tasks. This is the reason why theorem 5 computes the number of intervals in which the candidate task can be scheduled according to this first set of task. Then, we form with the reminder already proved schedulable tasks a number of sets, each set satisfy the previous property with a $(GCD \neq \boldsymbol{BG})$ and each set has a different GCD. For each set (excepting the first set) theorem 6 allows us to compute the number of intervals where the candidate task cannot be scheduled for non schedulability with one or several tasks from this set. As these intervals are included in the first calculation, we need to subtract them from the result of theorem 5. If the final result is less or equal to zero then there are no intervals on which the candidate task can be executed ⊡

Notice that, throughout this schedulability analysis, in the given conditions nothing is mentioned about precedence constraints, whereas we allow it in the tasks model. Indeed, a system with precedence constraints but without any periodicity constraint is always schedulable. The following theorem demonstrates that for a set of proved schedulable tasks i.e. satisfying periodicity constraints, it always exists a scheduling of these tasks which satisfies precedences between them tasks whatever precedences are.

Theorem 7 *Let Ω be a set of proved schedulable tasks. Whatever precedence constraints between Ω's tasks are, it exists, at least one scheduling which satisfies these precedence constraints*

Proof
Once a set of n tasks is proved schedulable, these tasks

can be scheduled in $n!$ different ways or orders. From these $n!$ orders, at least, one order satisfies the precedence constraints (we remind that tasks are not allowed to be preempted)$\boxdot$

5 Application

In order to illustrate the proposed method we propose the following example: let $(X : E(X) = 2, T(X) = 30)$ the candidate task to the schedulability analysis. Let Ω=$\{(A : 2, 5), (B : 1, 10), (C : 1, 20), (D : 1, 30), (E : 1, 60)\}$ the set of tasks already proved schedulable. In this case $\boldsymbol{BG}$=GCD$(T(A), T(B), T(C), T(D)) = 5$

The different subsets that we can set up from the set Ω and task X are:

- $\{X, A\}$ with the GCD of periods equal to 5
- $\{X, B, C\}$ with the GCD of periods equal to 10
- $\{X, D, E\}$ with the GCD of periods equal to 30

According to the corollary A is equal to: $\frac{30}{5}\left\lfloor\frac{5-2}{2}\right\rfloor =$ 6

and B is equal to: $B_1 + B_2$

$B_1 = (\frac{30}{10}(\left\lfloor\frac{5-1}{2}\right\rfloor\left\lfloor\frac{2}{3}\right\rfloor + (\left\lfloor\frac{3}{2}\right\rfloor - \left\lfloor\frac{2-2}{2}\right\rfloor))) =$
$3 \times 1 = 3$

and by the same way $B_2 = 1$, so $B =$
$3 + 1 = 4$

then $A - B = 6 - 4 = 2$, from this, we deduce that X is schedulable.

6 Conclusion

This paper provides a schedulability analysis in the case of non-preemptive tasks under strict periodicity and precedence constraints. Unlike previous research the periodicity constraint we deal with is strict which means that the elapsed time between two successive task execution is always equal to the period of this task.

The analysis proposed here is composed in several stages to reach the main result which is a necessary and sufficient condition we obtain at the end through a corollary. This condition allows us to check if a task can be scheduled or not.

Such schedulability analysis can be used in suboptimal heuristics to find an assignment of the tasks for each processor when partitioned multiprocessor scheduling is intended.

References

[1] L. Cucu and Y. Sorel. Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints. In *Proceedings of the 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG'04*, Cork, Ireland, December 2004.

[2] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12 th IEEE Symposium on Real-Time Systems*.

[3] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. *IEEE Design and Test of Computers*, 15(1):71–82, 1998.

[4] K. Danne and M. Platzner. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 568–573, August 2005.

[5] M. Alfano, A. Di-Stefano, L. Lo-Bello, O. Mirabella, and J.H. Stewman. An expert system for planning real-time distributed task allocation. In *Proceedings of the Florida AI Research Symposium*, Key West, FL, USA, May 1996.

[6] T. F. Abdelzaher and K. G. Shin. Period-based load partitioning and assignment for large real-time applications. *IEEE Transactions on Computers*, 49 (1):81–87, January 2000.

[7] O. Kermia and Y. Sorel. A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. In *Proceedings of ISCA 20th international conference on Parallel and Distributed Computing Systems, PDCS'07*, Las Vegas, Nevada, USA, September 2007.

[8] C. L.Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[9] L. George, P. Muhlethaler, and N. Rivierre. Optimality and non-preemptive real-time scheduling revisited. *Rapport de Recherche RR-2516, INRIA*, 1995.

[10] D. Kirby. On bezout's theorem. *The Quarterly Journal of Mathematics*, Dec 1988.

[11] S. K. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1-2):9–20, 2006.