2013 IEEE 16th International Conference on Computational Science and Engineering

Monoprocessor Real-Time Scheduling of Data Dependent Tasks with Exact Preemption Cost for Embedded Systems

Falou Ndoye
INRIA Paris-Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
falou.ndoye@inria.fr

Yves Sorel
INRIA Paris-Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
yves.sorel@inria.fr

***Abstract*—Most safety critical embedded systems, i.e. systems for which constraints must necessarily be satisfied in order to avoid catastrophic consequences, consist of a set of data dependent tasks which exchange data. Although non-preemptive real-time scheduling is safer than preemptive real-time scheduling in a safety critical context, preemptive real-time scheduling provides a better success ratio, but the preemption has a cost. In this paper we propose a schedulability analysis for data dependent periodic tasks which takes into account the exact preemption cost, data dependence constraints without loss of data and mutual exclusion constraints.**

Keywords-real-time scheduling, exact preemption cost, data dependent tasks, data transfer, mutual exclusion constraints, schedulability analysis, safety critical embedded systems.

I. INTRODUCTION

We focus on safety critical embedded systems, i.e. systems for which constraints must necessarily be satisfied in order to avoid catastrophic consequences. Such systems, in most cases, consist of a set of dependent periodic tasks resulting from a functional specification, usually performed with tools such as Simulink [1], Scade [2], etc., based on block diagrams. The functional specification describes the functions that will be executed and their dependences which represent the data produced and consumed by the functions. Such dependences involve a precedence relation on the execution of every producer function and one or several consumer functions, and lead to sharing the data considered. Dependent functions associated with temporal characteristics become dependent real-time tasks. These characteristics feature first release, Worst Case Execution Time (WCET), period and deadline.

Although non-preemptive real-time scheduling is safer than preemptive real-time scheduling in a safety critical context, preemptive real-time scheduling provides a better success ratio, but the preemption has a cost. That cost corresponds to the duration to save the context of the preempted task and the duration to restore this context when the preempted task will be seleceted again for execution. Due to its cost, a preemption increases the response time of the preempted task that may cause another preemption, and so on, leading to the avalanche phenomenon. The cost of the preemption is usually approximated in the WCET as assumed, explicitly, by Liu and Layland in their pioneering article [3]. However, such an approximation is dangerous in a safety critical context since a system could miss some deadlines during its real-time execution even though the schedulability conditions have been satisfied or, in the best case, resources could be wasted when the preemption cost is approximated through margins added to the WCET. This is why it is important to take into consideration the exact preemption cost.

The remainder of the paper is organized as follows: Section II describes the model and the notations used. Section III presents related work on dependence constraints, mutual exclusion constraints and the preemption cost. Section IV presents the interval used to achieve the schedulability analysis. Section V presents the data transfer involved by data dependences. Section VI presents the mutual exclusion constraints due to the shared buffers containing the data. Section VII presents the schedulability analysis we propose. Section VIII presents the impact of the preemption cost in the schedulability analysis. Finally, Section IX concludes and gives some directions for future work.

II. MODEL AND NOTATIONS

A. Model

Let $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of n preemptive, dependent real-time tasks represented by a directed acyclic graph (DAG) noticed G. The nodes of G represent the tasks and the arcs (directed edges) correspond to the dependences between the tasks. An example of a graph depicting dependent tasks is given in Figure 1.

Every periodic task is denoted by $\tau_i = (r_i^1, C_i, D_i, T_i)$ where r_i^1, C_i, D_i and T_i are the temporal characteristics of the task. r_i^1 is the first release time, C_i is the WCET without any added time to approximate the preemption cost, D_i is the relative deadline, and T_i the period of the task τ_i. We assume that $C_i \leq D_i \leq T_i$. Actually, a periodic task executes indefinitely a "job" also called an "instance". We

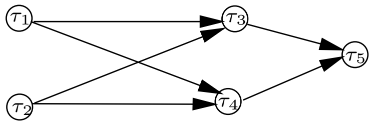

Figure 1. Example of DAG representing a dependent set of tasks

assume that Γ_n is scheduled off-line and the priorities of the tasks are assigned according to a fixed-priority scheduling algorithm denoted by $Algo$. This off-line schedule is stored in a table that will be used during the real-time execution of the tasks. This approach avoids the variable cost of the overhead caused by an on-line scheduling algorithm. We assume that the dependent tasks must have the same period or have multiple periods in order for the consumer task to be able to receive all the data produced by the producer task without any data being lost or duplicated. Note that, when the cost of every data to transfer is taken into account, this is the worst situation since the other situations lead to less data transfers. We assume that each producer task τ_i has a buffer b_i in which it writes the data at the end of its execution. On the other hand a consumer task τ_j may read this data in this buffer b_i during its execution. If τ_j is preempted by τ_i during its execution, the task τ_i may write a data in the buffer b_i while after τ_j will resume it may read the buffer b_i with the wrong value. This is why buffer b_i must not be read by τ_i and also written by τ_j. This restriction is called a mutual exclusion constraint on b_i between τ_i and τ_j. Note however that, several tasks can read simultaneously in any buffer.

Finally, we assume that the processor where the tasks will be executed, has neither cache nor pipeline, or complex internal architecture. Both previous assumptions are usually made in safety critical embedded systems where determinism is a key issue.

B. Notations

We introduce some notations in Table I.

Notation	Description
$H_n = LCM\{T_i\}_{1\leq i\leq n}$	The hyper period of Γ_n, the least common multiple of the periods
$r_{min} = min\{r_i^1\}_{1\leq i\leq n}$	The minimum of the first release time of the tasks in Γ_n
$r_{max} = max\{r_i^1\}_{1\leq i\leq n}$	The maximum of the first release time of the tasks in Γ_n
$pred(\tau_i)$	Set of predecessors of τ_i
$succ(\tau_i)$	Set of successors of τ_i
$b_i(t)$	The number of data produced by τ_i and available in the buffer b_i at t
$I(t)$	The set of release and end execution times at t, in the increasing order
$r^-(t)$	The preview time in $I(t)$ from t
$r^+(t)$	The next time in $I(t)$ from t
$\Gamma_r(t)$	The set of ready tasks at time t
$\phi_i(t)$	The task selected for execution at time t among the ready tasks $\Gamma_r(t)$
$c_i(t)$	The remaining execution time of τ_i at time t
$d_i(t)$	The deadline of τ_i relative to time t
p_i	The fixed priority of τ_i according to $Algo$
p_{b_i}	The priority of buffer b_i
$\Gamma(\tau_i, t)$	The set of tasks blocked by τ_i at t
$\Gamma(b_i)$	The set of tasks that may use buffer b_i
$B(t)$	The set of buffers that are currently used at t
$\Gamma(B(t))$	The set of tasks that currently use the buffers in $B(t)$

Table I
SUMMARY OF THE NOTATIONS

III. Related work and contributions

A. Data Dependence constraints

A dependence constraint between two tasks (τ_i, τ_j) can be: $1)$ a precedence constraint [4], i.e. task τ_i must be executed before τ_j. Task τ_i is called the predecessor task and τ_j the successor task; $2)$ a data dependence constraint [5], i.e. τ_i must be executed before τ_j and τ_j (consumer task) must wait to receive the data from τ_i (producer task) before its execution.

There are two kinds of precedence constraints: simple precedence constraints [4], [6], [7], i.e. each execution of τ_j is preceded by an execution of τ_i and extended precedence constraints [8], [7], i.e. the tasks do not have the same periods and are executed at different rate. There are two approaches to dealing with the precedence constraints. The first approach is based on semaphores [9]: a semaphore is allocated for each precedence (τ_i, τ_j), and the successor task τ_j must wait for the predecessor task τ_i to release the semaphore before it can start its execution. The second approach is based on the modification of the priorities and the release times of the task [4], [7].

The data dependence constraints require that the precedence constraints are satisfied. In the systems considered, since the tasks can have different execution periods, a consumer task can consume the first data, the last, all the data, or any other combinations of data from a producer task. In [8], when the consumer task has a period greater than or equal to that of the producer task, then the data consumed is the last data produced during the execution of the consumer. The authors transform the extended precedences constraint into a simple precedence constraint by replacing a task τ_i by $n_i = \frac{H_n}{T_i}$ duplicated tasks of period H_n (H_n is the least common multiple of the periods of the tasks and T_i is the period of τ_i). They use the principle proposed in [4] to deal with the simple precedence constraints. This method based on the duplication of some tasks is exponential in time and memory. In [5], 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 the data from the n executions of the producer task (the data produced differ from one execution of the producer task to another execution and 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. In order to do that, the graph of dependent tasks is unrolled. In that study the authors consider that tasks are non-preemptive and have strict periods.

B. Mutual exclusion constraints

In order to satisfy mutual exclusion constraints, the dependent tasks are executed in different critical sections. These critical sections must be managed carefully, because they can cause priority inversion which occurs when a lower priority task blocks the execution of a higher priority task. Many synchronization protocols have been proposed to limit the blocking duration of a task. Among them, the priority inheritance protocol (PIP) [10] based on the principle of priority inheritance. The PIP has two problems, first the blocking duration can be long, and second it does not prevent deadlocks. A deadlock occurs when at time t_1, a task τ_j starts to use a buffer b in a critical section until at time t_2 it attempts to use another buffer b', however at this time, the higher priority task τ_i preempts it and begins to use buffer b' in a critical section. To reduce the blocking duration and prevent deadlocks, the priority ceiling protocol (PCP) was proposed in [10]. The principle of PCP is to ensure that when a task τ_i preempts another task which is executing in a critical section and τ_i executes its own critical section, the priority of τ_i is guaranteed to be higher than the inherited priorities of all the preempted tasks in critical sections. If this condition cannot be satisfied, τ_i cannot start its critical section and it is suspended. When τ_i is blocked by another task, the latter inherits the priority of τ_i. The PIP and PCP protocols are used with static priority algorithms. For dynamic priority algorithms, the stack resource policy (SRP) [11] is proposed. The SRP is a dynamic version of PCP.

C. Exact preemption cost

For preemptive scheduling without data-dependence, there have been very few studies that address the exact number of preemptions. Of those that have, A. Burns, K. Tindell and A. Wellings in [12] presented an analysis that enables the global cost due to preemptions to be factored into the standard equations for calculating the worst case response time of any task, but they achieved that by considering the maximum number of preemptions rather than the exact number. Juan Echagüe, I. Ripoll and A. Crespo also tried to solve the problem of the exact number of preemptions in [13] by computing the schedule using idle times and counting the number of preemptions. However, they did not really determine the execution overhead incurred by the system due to these preemptions, as they did not take into account the cost of each preemption during the analysis. Hence, this amounts to considering only the minimum number of preemptions because some preemptions are not considered i.e., those due to the increase in the execution time of the task because of the cost of preemptions themselves. To our best knowledge the only studies in which the exact preemption cost is taken into account in the schedulability analysis are those presented in [14]. The authors proposed a scheduling operation named $\oplus$ that performs a schedulability analysis while computing the exact number of preemptions. However, this operation does not allow priority inversion since the priorities of the tasks cannot change during their execution.

IV. Schedulability interval

A schedulability interval is the finite interval such that the computation of a schedule during this interval provides enough information to define the infinite schedule.

Goossens and Devillers in [15], considering independent tasks and fixed-priority scheduling algorithms, determined a schedulability interval. On the other hand, Leung and Merrill in [16] showed that the interval $[r_{min}, r_{max}+2*H_n]$ is a schedulability interval for a set of n independent periodic tasks for the EDF (Earliest Deadline First) scheduling algorithm. This interval is not necessarily the shortest. Then, in [17], Choquet and Grolleau proposed a minimal schedulability interval for a set of n dependent periodic tasks. The schedulability interval proposed by those authors denoted by I_n is given by:

$$I_n = [r_{min}, t_c + H_n] \tag{1}$$

where t_c is the time from which the schedule repeats indefinitely. t_c is computed iteratively by an algorithm given in the article. Since t_c is smaller or equal to $r_{max} + H_n$, we will use in the following the interval given by the Equation 1 with $tc = r_{max} + H_n$.

V. Data transfer

We consider a pair of data dependent tasks (τ_i, τ_j) $\in G$ involving a data transfer where task τ_i produces data for τ_j which consumes it. If $T_i = T_j$ then task τ_i is executed once before one execution of τ_j, that is equivalent to a simple precedence constraint. If it is not the case, according to their periods, there are two possibilities:

- if $T_i < T_j$ then task τ_i is executed $k_{ij} = \frac{T_j}{T_i}$ times before one execution of τ_j. During its execution, task τ_j consumes all the k_{ij} data produced by τ_i. As mentioned above, when $T_i \neq T_j$, there are several cases for τ_j to consume the data produced by τ_i. As in [5], we suppose that τ_j consumes all the data produced by τ_i without any loss of data, since it is the worst case;

- if $T_i > T_j$ then task τ_i is executed once before $k_{ji} = \frac{T_i}{T_j}$ executions of τ_j. The task τ_j consumes one data produced by τ_i for each of its executions.

Since we have supposed that T_i and T_j are multiple, both cases can be summarized in one case given by: task τ_i is executed $k_{ij} = \lceil \frac{T_j}{T_i} \rceil$ times before $k_{ji} = \lceil \frac{T_i}{T_j} \rceil$ executions of τ_j.

Definition 1: A data transfer between a task τ_j that consumes data produced by a task τ_i at time t, $t \in I_n$ and $t \geq r_j^1$, is said to be without loss of data if and only if:

- τ_j receives, or has already received, all the k_{ij} data produced by τ_i, $\forall \tau_i \in pred(\tau_j)$,
- τ_j has executed less than k_{ji} times after receiving the data from τ_i, $\forall \tau_i \in pred(\tau_j)$,
- τ_j does not produce more than k_{ji} data for each $\tau_i \in succ(\tau_i)$,
- all the k_{ji} data already produced by τ_j for τ_i, $\forall \tau_i \in succ(\tau_i)$ must be already consumed k_{ij} times by τ_i.

Figure 2 presents an example of three data dependent tasks which satisfy Definition 1. In this figure, task $\tau_1 = (0, 2, 6, 6)$ produces data for task $\tau_2 = (1, 1, 3, 3)$ which produces data for task $\tau_3 = (2, 1, 6, 6)$. Task τ_1 is executed $k_{12} = \lceil \frac{T_2}{T_1} \rceil = \lceil \frac{3}{6} \rceil = 1$ time before $k_{21} = \lceil \frac{T_1}{T_2} \rceil = \lceil \frac{6}{3} \rceil = 2$ executions of task τ_1. Task τ_2 consumes exactly $k_{21} = 2$ times the same data produced by τ_1. Also task τ_2 is executed $k_{23} = \lceil \frac{T_3}{T_2} \rceil = \lceil \frac{6}{3} \rceil = 2$ times before $k_{32} = \lceil \frac{T_2}{T_3} \rceil = \lceil \frac{3}{6} \rceil = 1$ execution of task τ_3. Thus, task τ_3 consumes $k_{23} = 2$ data from τ_1 for each of its executions.

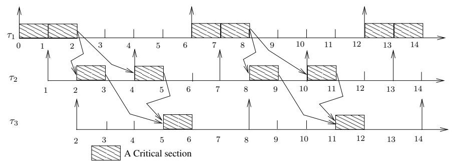

Figure 2. Data transfer

We define by $b_i : I(t) \rightarrow \mathbb{N}$ the function which determines the number of data produced by τ_i until time t. Before a task τ_i finishes its first execution ($t < r_i^1 + C_i$), $b_i(t)$ is initialized to zero because no data is produced at this time. If τ_i produces a data during its execution at $r^-(t)$ then $b_i(t) = b_i(r^-(t)) + 1$ else $b_i(t) = b_i(r^-(t))$.

$\forall t \in I(t)$, $b_i(t)$ is given by:

$$b_i(t) = \begin{cases} 0 & \text{if } (t < r_i^1 + C_i) \\ b_i(r^-(t)) + 1 & \text{if } ((\phi(r^-(t)) = \tau_i) \wedge \\ & \quad (r^-(t) + c_i(r^-(t)) \leq t)) \text{ else} \\ b_i(r^-(t)) & \end{cases}$$

The remaining execution time at time t for a task τ_i denoted by $c_i(t)$ and the selected task at t denoted by $\phi(t)$ are presented in the section VII.

Definition 2: A task τ_i is ready for execution at $t \in I(t)$ if and only if:

1) $t \geq r_i^1$;
2) $(r^-(t) + c_i(r^-(t)) > t) || ((c_i(r^-(t)) > 0) \wedge (\phi(r^-(t)) \neq \tau_i))$;
3) $b_j(t) * k_{ij} - b_i(t) * k_{ji} \geq k_{ji}\ \forall \tau_j \in pred(\tau_i)$;
4) $b_i(t) * k_{ji} - b_j(t) * k_{ij} < k_{ij}\ \forall \tau_j \in succ(\tau_i)$.

Conditions 1) and 2) ensure that τ_i is released before, or at, t and τ_i does not complete its execution. Condition 3) ensures that task τ_i waits to receive all the k_{ji} data produced by each of these predecessors τ_j and that τ_i is not executed more than k_{ij} times after receiving the k_{ji} data from τ_j. Condition 4) ensures that no more than k_{ij} data are produced by τ_i before its successor τ_j consumes these data.

Definition 3: The set of ready tasks at time t denoted by $\Gamma_r(t)$ is given by: $\Gamma_r(t) = \{\tau_i \in \Gamma_n / \tau_i$ is ready at t, according to Definition 2$\}$

VI. Mutual exclusion constraints

Since we consider a fixed-priority scheduling algorithm, we use the priority ceiling protocol to deal with the mutual exclusion constraints on the shared buffers. A priority ceiling is assigned to each shared buffer which is equal to the highest priority task that may use this resource. The priority ceiling of the buffer b_i is given by

$$p(b_i) = max\{p_i\}_{\tau_i \in \Gamma(b_i)}$$

Since a task inherits the highest priority of the tasks that blocks it , this means that the priority of that considered task may change during its execution. We denote by $p_i(t)$ the priority of a task τ_i at time t given by Equation 2.

$$p_i(t) = \begin{cases} max\{p_j(t)\}_{\tau_j \in \Gamma(\tau_i, t)} \text{ if } \Gamma(\tau_i, t) \neq \emptyset \text{ else} \\ p_i \end{cases} \quad (2)$$

We denote by $p(t)$ the highest priority ceiling of the buffers which are being used at time t. $p(t)$ is given by Equation 3.

$$p(t) = max\{p(bi)\}_{b_i \in B(t)} \quad (3)$$

Property 1: At time $t \in I_n$, a task $\tau_i \in \Gamma_r(t)$ is allowed to start a new critical section if and only if the priority of τ_i at t is higher than all the priority ceilings of the buffers which are being used by tasks other than τ_i:

$$p_i(t) > p(t)$$

VII. Schedulability analysis

We consider a set of dependent tasks Γ_n according to the model presented in Section II-A. The schedulability analysis of Γ_n is achieved in the schedulability interval I_n (see Section IV). We define a selection function denoted by $\phi : I(t) \rightarrow \Gamma_r(t)$ which determines the ready task which executes at t. As for a scheduler, ϕ is only defined at the release and end execution times of the tasks. We use this selection function to determine the remaining execution time of a task τ_i given by the function $c_i : I(t) \rightarrow \mathbb{N}$. We will need also the deadline of τ_i relative to t, determined by the function $d : I(t) \rightarrow \mathbb{N}$.

We denote by $F = \{t \in I_n / \exists(\tau_i, k) \in (\Gamma_n, \mathbb{N}), t = r_i^1 + kT_i\}$, the set of the release times in I_n of tasks in Γ_n. The set of release and end execution times of all the tasks at time t denoted by $I(t)$ is computed by:

$$I(t) = \begin{cases} F \cup \{t + c_k(t)\} \text{ if } ((t = r_{min}) \wedge (\phi(t) = \tau_k) \wedge \\ \qquad (t + c_k(t) < r^+(t))) \text{ else} \\ I(r^-(t)) \cup \{t + c_k(t)\} \text{ if } ((\phi(t) = \tau_k) \wedge \\ \qquad (t + c_k(t) < r^+(t)) \text{ else} \\ I(r^-(t)) \end{cases}$$

Since we achieve an off-line scheduling, we determine the end execution times of the tasks according to their worst execution time and not according to their execution time at runtime. We assume that, at runtime, the scheduler selects a task for execution only at the times defined in the scheduling table provided afer the off-line scheduling.

A. Selected task for execution at t

ϕ is the function which determines the task selected for execution at time $t \in I(t)$. At $t = r_{min}$, the task selected is the one which has no predecessor and which has the highest priority p_i. But if $t > r_{min}$ then $\phi(t)$ is determined according to the schedule of the system until t. In order to ensure the data transfer is compliant with Definition 1, the selected task $\phi(t)$ must be in the set of ready tasks $\Gamma_r(t)$ defined in Section V. Also, in order to avoid deadlock and to minimize the blocking duration according to PCP, the selected task must satisfy Property 1.

$\forall t \in I(t)$, $\phi(t) = \tau_k \in \Gamma_r(t)$ /

$$\begin{cases} (p_k(t) = max\{p_i(t)_{\tau_i \in \Gamma_r(t)}\}) \wedge (B(t) = \{\emptyset\}) \text{ else} \\ p_k(t) > p(t) \text{ else} \\ p_k(t) = max\{p_i(t)_{\tau_i \in \Gamma(B(t))}\} \end{cases}$$

At t, if no task is selected for execution, then the processor is idle, denoted by $\phi(t) = idle$. When task τ_i is the selected task at t, τ_i is executed until $r^+(t)$ corresponding to the end execution of τ_i or a release time of a task in Γ_n.

B. Remaining execution time of a task at t

We denote by c_i the function which determines, at each time t, the remaining execution time of the tasks τ_i. This remaining time corresponds to the number of time units that τ_i must execute to complete its execution in the instance $l = \lceil \frac{t}{T_i} \rceil$. For the computation of $c_i(t)$, we take into account the exact preemption cost. That is, at each time $t \in I(t)$, if τ_i is preempted by another task, then the cost associated to one preemption denoted by α is added to the number of time units that τ_i must execute to complete its execution. This principle allows us to take into account the preemptions caused by other preemptions.

At each time t, the task τ_i can be in different states and $c_i(t)$ is computed according to these states.

- if τ_i is released at t $((\frac{t-r_i^1}{T_i}) \in \mathbb{N})$ then $c_i(t) = C_i$ else,
- if τ_i is not executed at $r^-(t)$ meaning that τ_i is not selected at $r^-(t)$ $(\phi(r^-(t)) \neq \tau_i)$ then $c_i(t) = c_i(r^-(t))$ else,
- if τ_i is executed at $r^-(t)$ and is not preempted at t then $c_i(t) = (r^-(t) + c_i(r^-(t)) - t)$ else,
- τ_i is executed at $r^-(t)$ and is preempted at t then $c_i(t) = (r^-(t) + c_i(r^-(t)) - t) + \alpha$, with α being the cost of one preemption.

$\forall t \in I(t)/t \geq r_i^1$, $c_i(t)$ is given by:

$$c_i(t) = \begin{cases} C_i & \text{if } (\frac{t-r_i^1}{T_i}) \in \mathbb{N} \text{ else} \\ c_i(r^-(t)) & \text{if } (\phi(r^-(t)) \neq \tau_i) \text{ else} \\ r^-(t) + c_i(r^-(t)) - t & \text{if } (\phi(t) = \tau_i) \vee \\ \qquad ((\phi(t) \neq \tau_i) \wedge (r^-(t) + c_i(r^-(t)) = t)) \text{ else} \\ (r^-(t) + c_i(r^-(t)) - t) + \alpha \end{cases}$$

C. Deadline of a task at t

We denote by d_i the function which determines, at each time t the deadline of τ_i. This means that at time t the task τ_i has to complete its execution no later than $t + d_i(t)$ otherwise task τ_i will miss its deadline. For each release time of τ_i $(\forall t/(\frac{t-r_i^1}{T_i}) \in \mathbb{N})$ $d_i(t) = D_i$. This deadline of task τ_i at time t decreases until zero, i.e. when the deadline D_i is reached.

$\forall t \in I(t) \geq r_i^1$, $d_i(t)$ is given by:

$$d_i(t) = \begin{cases} D_i & \text{if } (\frac{t-r_i^1}{T_i}) \in \mathbb{N} \text{ else} \\ r^-(t) + d_i(r^-(t)) - t & \text{if } r^-(t) + d_i(r^-(t)) > t \text{ else} \\ 0 \end{cases}$$

D. Schedulability analysis

Theorem 1: A task $\tau_i \in \Gamma_n$ is schedulable if and only if:

$$\begin{array}{l} \forall t \in I(t), \ (c_i(t) \leq d_i(t)) \wedge \\ ((t \leq r_i^1) \vee (c_i(r^-(t)) = 0) \vee \\ (\phi(r^-(t)) = \tau_i) \vee ((t - r_i^1) mod T_i \neq 0)) \end{array} \quad (4)$$

Proof: Suppose that the condition 4 is false, then we have:

$$\exists t \in I(t),\ ((c_i(t) > d_i(t)) \vee$$
$$((t > r_i^1) \wedge (c_i(r^-(t)) > 0) \wedge$$
$$(\phi(r^-(t)) \neq \tau_i) \wedge ((t - r_i^1) mod T_i = 0))$$

If $c_i(t) > d_i(t))$ then task τ_i cannot finish its execution before $t + d_i(t)$ and will miss its deadline at $t + d_i(t)$. In this case τ_i is not schedulable. If, $((t > r_i^1) \wedge (\phi(r^-(t)) \neq \tau_i) \wedge (c_i(r^-(t)) > 0) \wedge ((t - r_i^1) mod T_i = 0))$ means that at time $r^-(t)$, τ_i is not selected for execution and τ_i is already released. Since $((t - r_i^1) mod T_i = 0)$, task τ_i begins a new instance at time t while it did not finish its execution during its last instance $(c_i(r^-(t)) > 0)$, then τ_i misses its deadline at time t, therefore τ_i is not schedulable. Thus if the condition 4 is false then τ_i is not schedulable t. If $\tau_i \in \Gamma_n$ is schedulable then the condition 4 is true.

Suppose that the condition 4 is true. $\forall t \in I(t),\ c_i(t) \leq d_i(t)$. If $(c_i(r^-(t)) = 0) \vee (t \leq r_i^1)$ then τ_i has finished its execution at $r^-(t)$, or τ_i is not already released at t or it is released released at t. In all the cases τ_i is schedulable at t. If $\phi(r^-(t)) = \tau_i$ then τ_i is executed at $r^-(t)$ and $c_i(t) \leq d_i(t))$ then τ_i is schedulable at t. If $(t - r_i^1) mod T_i \neq 0$ then τ_i is always schedulable in its current instance then τ_i is shedulable at t. Thus if the condition 4 is true then τ_i is schedulable at any time $t \in I(t)$ therefore τ_i is schedulable. ■

Definition 4: The set of dependent periodic tasks Γ_n is schedulable while taking into account the exact preemption cost, if $\forall \tau_i \in \Gamma_n$ then τ_i is schedulable according to the theorem 1.

The schedulability analysis is achieved by the algorithm 1.

Algorithm 1 Schedulability analysis

```
1: $t = r_{min}$
2: $schedulable \leftarrow true$
3: **while** $(t < (tc + H_n)) \wedge (schedulable = true)$ **do**
4:    compute $\phi(t)$
5:    Compute $I(t)$
6:    **for** i=1 to n **do**
7:       **if** $(t \geq r_i^1)$ **then**
8:          compute $c_i(t)$
9:          compute $d_i(t)$
10:         **if** $((c_i(t) > d_i(t)) \vee ((t > r_i^1) \wedge (c_i(r^-(t)) > 0) \wedge (\phi(r^-(t)) \neq \tau_i) \wedge ((t - r_i^1) mod T_i = 0))$ **then**
11:            $schedulable \leftarrow false$
12:         **end if**
13:      **end if**
14:   **end for**
15:   $t \leftarrow r^+(t)$
16: **end while**
```

The complexity of this schedulability analysis in the worst case is equal to $O(2.m.n)$, with m the number of different release times in I_n.

Theorem 2: $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ is a set of dependent periodic tasks.

If Γ_n is schedulable according to Algorithm 1, while taking into account the exact preemption cost then, Γ_n remains schedulable when a task τ_i has at runtime an execution time smaller than its worst case execution time.

Proof: Suppose that Γ_n is schedulable according to Algorithm 1, while taking into account the exact preemption and T is its scheduling table provided after the off-line scheduling. At runtime, if a task τ_i has an execution time smaller than its worst case execution time then it completes its execution early than the time defined in T. Task τ_i remains schedulable because it does not miss its deadline. This fact does not affect the schedulability of the other tasks because each task start its execution exactly at the times defined in the table T. This involved that a task can complete early its execution but will not miss its execution, then the tasks in Γ_n remain schedulable. Thus if Γ_n is schedulable according to Algorithm 1, while taking into account the exact preemption cost then, Γ_n remains schedulable at runtime when a task τ_i has an execution time smaller than its worst case execution time. ■

Therefore the proposed off-line scheduling algorithm is sustainable [18].

E. Example

We consider the set of dependent tasks Γ_3 represented by graph G' in Figure 3. According to the data transfer presented in Section V, τ_3 must consume, for each of its executions, two data from τ_1 and one data from τ_2. The data produced by τ_1 are written in its buffer b_1 and those of τ_2 are written in its buffer b_2. For its execution, task τ_3 will read in the buffers b_1 and b_2 to consume the data produced by τ_1 and τ_2. We assume that the cost of one preemption denoted by α is equal to 1 time unit and the priorities of the tasks are assigned according to rate monotonic (RM) [3] fixed priority scheduling algorithm. The priority of a task τ_i is given by: $p_i = \frac{1}{T_i}$.

The hyperperiod of Γ_3 is $H_3 = 24$. The schedulability analysis of Γ_3 is achieved in the interval $I_3 = [r_2^1, t_c + H_3] = [r_2^1, r_3^1 + 2 * H_3] = [0, 58]$, with $t_c = r_3^1 + H_3$ as presented in Section IV. According to Algorithm 1, we obtain the scheduling table of Γ_3 presented in table II.

In the Table II, the symbols '-' means that task τ_1 and τ_3 are not released at t. At $t = 13, 16, 41$, no task is selected for execution, at these times the processor is idle. We only consider the released and end execution times of tasks in Γ_3, that allows a reduction of the complexity of the schedulability analysis.

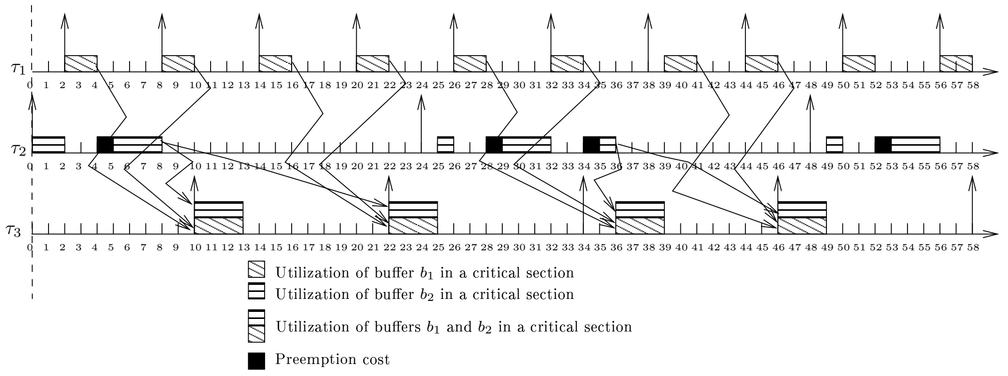
τ_1
τ_2
τ_3
Utilization of buffer b_1 in a critical section
Utilization of buffer b_2 in a critical section
Utilization of buffers b_1 and b_2 in a critical section
Preemption cost

Figure 4. Scheduling of the Γ_3 with exact preemption cost

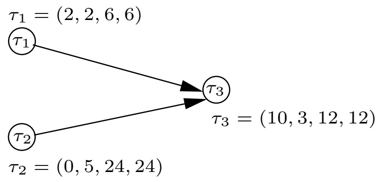

Figure 3. Graph G' representing Γ_3

t	$\phi(t)$	τ_1		τ_2		τ_3	
		$c_1(t)$	$d_1(t)$	$c_2(t)$	$d_2(t)$	$c_3(t)$	$d_3(t)$
0	τ_2	–	–	5	24	–	–
2	τ_1	2	6	4	22	–	–
4	τ_2	0	4	4	20	–	–
8	τ_1	2	6	0	16	–	–
10	τ_3	0	4	0	14	3	12
13	*idle*	0	2	0	11	0	9
14	τ_1	2	6	0	10	0	8
16	*idle*	0	4	0	8	0	6
20	τ_1	2	6	0	4	0	2
22	τ_3	0	4	0	2	3	12
24	τ_3	0	2	5	24	1	10
25	τ_2	0	1	5	23	0	9
26	τ_1	2	6	5	22	0	8
28	τ_2	0	4	5	20	0	6
32	τ_1	2	6	2	16	0	2
34	τ_2	0	4	2	14	3	12
36	τ_3	0	2	0	12	3	10
38	τ_3	2	6	0	10	1	8
39	τ_1	2	5	0	9	0	7
41	*idle*	0	3	0	7	0	5
44	τ_1	2	6	0	4	0	2
46	τ_3	0	4	0	2	3	12
48	τ_3	0	2	5	24	1	10
49	τ_2	0	1	5	23	0	9
50	τ_1	2	6	5	22	0	8
52	τ_2	0	4	5	20	0	6
56	τ_1	2	6	2	16	0	2

Table II
SCHEDULABILITY ANALYSIS

Figure 4 depicts the scheduling table II. In this figure, the exact preemption cost is taken into account. Task τ_2 is preempted at $t = 2, 26, 32, 50, 56$. The preemption of τ_2 at $t = 32$ is involved by its preemption occurred at $t = 26$, because if task τ_2 was not preempted at $t = 26$ or the cost of one preemption was supposed equal to 0 then τ_2 would not be preempted at $t = 32$. We observe also that the data transfer mechanism presented in Section V and the mutual exclusion constraints are respected. Task τ_3 consumes for each of its execution exactly 2 data produced by τ_1 and 1 data produced by τ_2. No data is lost. At $t = 38$, even if τ_1 has a higher priority than task τ_3, that is that latter which is selected for execution because τ_1 is blocked by τ_3 which uses the buffer b_1 at this time.

VIII. IMPACT OF THE PREEMPTION COST

In this section we present the impact of the preemption cost in the schedulability analysis. For the sake of simplicity we implement Algorithm 1 with independent tasks, by considering 2 cases. In the first case we suppose that the preemption cost is approximated in the WCET of the tasks ($\alpha = 0$). In the second case we suppose that the preemption cost is not approximated into the WCET with a cost $\alpha = 1$ for one preemption. We compare the success ratio for these 2 cases. We perform this comparison by testing 15 sets of task sets. Every set of task sets is composed of 10 task sets. Every of the latter task sets contains 10 tasks. In both cases, the Algorithm 1 is executed on every set of task sets. Then, we compute the success ratio and the average load of every set of task sets. The success ratio of a set of task sets, is defined by:

$$\frac{number\ of\ task\ sets\ schedulable}{number\ of\ task\ sets\ in\ a\ set\ of\ task\ sets}$$

In Figure 5, we observe that when the average load of the set of task sets is less than 0.8 then the scheduling without and with preemption cost has the same success ratio. But when this average load increases to 1 then the success ratio obtained by appying Algorithm 1 with preemption cost decreases until 0, while the success ratio of Algorithm 1

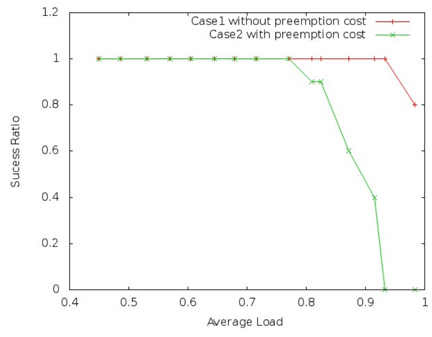

Figure 5. Impact of the preemption cost

without preemption cost is decreased until 0.8. When the average load is greater than 0.93, the success ratio of Algorithm 1 with preemption cost is equal to 0, that is, no task set is schedulable whereas in the case of Algorithm 1 without preemption cost some task sets are schedulable. The Figure 5 shows that more the load of the task sets is important, more it is dangerous to approximate the preemption cost in the WCET, because a task set could miss some deadlines during its real-time execution even though the schedulability conditions have been satisfied.

IX. Conclusion and future work

Considering fixed priority algorithms, we have presented a new schedulability analysis of data dependent periodic tasks which takes into account the exact preemption cost, without loss of data and mutual exclusion constraints which both involve priority inversions. The proposed off-line scheduling algorithm is sustainable, i.e. the set of dependent tasks remains schedulable at runtime when a task has an execution time smaller than its worst case execution time.

We plan to use this schedulability analysis in the case of partitioned and semi-partitioned multiprocessor real-time scheduling.

References

[1] The simulink tool page: http://www.mathworks.com/products/simulink/.

[2] The scade tool page: http//www.esterel-technologies.com/products/scade-suite/.

[3] C. L. Liu and J W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environnment. *Journal of the ACM*, vol. 20(1), January 1973.

[4] M. Silly H. Chetto and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time System.*, 2(3):181–194, sepember 1990.

[5] 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, sep 2007.

[6] M. Spuri and J. A. Stankovic. How to integrate precedence constraints and shared resources in real-time scheduling. *Computers, IEEE Transactions on*, 43(12):1407–1412, December 1994.

[7] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 301–310, Washington, DC, USA, 2010.

[8] P. Richard, Cottet, and M. Richard. On-line scheduling of real-time distributed computers with complex communication constraints. In *ICECCS*, pages 26–34, 2001.

[9] J. Forget, E. Grolleau, C. Pagetti, and P. Richard. Dynamic Priority Scheduling of Periodic Tasks with Extended Precedences. In *IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA)*, Toulouse, France, September 2011.

[10] R. Rajkumar L. Sha and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39:1175–1185, 1990.

[11] Theodore P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

[12] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 21:475–480, May 1995.

[13] J. Echague, I. Ripoll, and A. Crespo. Hard real-time preemptively scheduling with high context switch cost. In *Proceedings of 7th Euromicro workshop on Real-Time Systems*, Los Alamitos, CA, USA, 1995. IEEE Computer Society.

[14] P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, Pisa, Italy, July 2007.

[15] J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, 13(2):107–126, 1997.

[16] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3), November 1980.

[17] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theor. Comput. Sci.*, 310(1-3):117–134, 2004.

[18] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 159–168, Washington, DC, USA, 2006. IEEE Computer Society.