Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems

By: J.C. Palencia and M. González Harbour
Departamento de Electrónica y Computadores, Universidad de Cantabria, SPAIN

Abstract

In this paper we present improved techniques for the schedulability analysis of tasks with precedence relations in multiprocessor and distributed systems, scheduled under a preemptive fixed priority scheduler. Recently developed techniques, based on the analysis of tasks with dynamic offsets, take into account the precedence relations between tasks only indirectly, through terms iteratively estimated from the response times of the tasks. With the techniques presented in this paper, we exploit the precedence relations in a more accurate way, and we also take advantage of the priority structure of the different tasks. These considerations permit a significant improvement of the results of the analysis applied to distributed and multiprocessor systems.

1. Introduction

Rate monotonic analysis (RMA) [2][4] allows an exact calculation of the worst-case response time of tasks in single-processor real-time systems, including the effects of task synchronization [9], the presence of aperiodic tasks, the effects of deadlines before, at or after the periods of the tasks [3], precedence constraints and tasks with varying priorities [1], etc. However current RMA cannot provide exact solutions to the response times in multiprocessor and distributed hard real-time systems. The general worst-case analysis of a distributed system is still an open issue, but one approach that can be used is to analyze each processing and communication resource of the system as if it were independent. Tindell and Clark developed a widely accepted technique, based on the assumption that all tasks are independent [10][5] with a jitter term due to the execution of preceding tasks. In [7] we derived an approximate technique based on the model of tasks with dynamic offsets, which significantly improved the results obtained in systems with tasks that suspend themselves, and also in distributed systems. Nevertheless, these techniques are still somehow pessimistic since they do not exploit sufficiently the precedence relations among tasks, within the analysis. In this article we improve the analysis based on tasks with dynamic offsets, by directly including the precedence relations among tasks of the same response sequence in the analytical expressions. As we will show, in distributed systems the new technique allows a significant increase of the schedulable utilization of the CPU compared to the case when previous analysis techniques were used. This comes at no cost for the application, which will still be scheduled using fixed priorities.

The paper is organized as follows. In Section 2, we review the analysis technique derived from the model of tasks with dynamic offsets, which is directly applicable to distributed systems. In section 3, we present the activation conflicts, a direct consequence of the precedence relationships among tasks of the same response sequence, and we will see how we can take advantage of their existence to improve the analysis. In section 4, we derive analytical expressions leading to the calculation of the worst-case execution interference. Later, in section 5, we consider the effect of the precedence relationships on tasks belonging to the same sequence as the task under analysis, and which complete the set of equations applicable to the analysis. Section 6 shows the simulation results obtained with the new technique, comparing them with those of current techniques, based on independent tasks and dynamic offsets, respectively. Finally, in Section 7 we give our conclusions.

2. Analysis based on dynamic offsets

2.1. Computational Model

The system model that we will consider as an approximation to the distributed system is composed of a set of tasks executing in the same or different processors, which are grouped into entities that we will call *transactions* [11]. Each transaction Γ_i is activated by a periodic sequence of external events with period T_i, and contains a set of m_i *tasks*. The relative phasings between the different external events are arbitrary. Each task is activated (released) when a relative time —called the *offset*— elapses after the arrival of the external event. Each activation of a task releases the

This work has been supported in part by the *Comisión Interministerial de Ciencia y Tecnología* of the Spanish Government, under grant number TAP97-892

execution of one instance of that task, that we will call a *job*. Each task has its own unique priority, and the task set is scheduled using a preemptive fixed priority scheduler. When the activation of the task occurs with an offset that is constant, independent of the execution of other tasks in the system, we say that is a *static* offset. We call an offset *dynamic* if it can vary between some minimum and maximum interval. This variation is often caused by the execution of other tasks or activities for which the activated task must wait.

Each task is identified with two subscripts: the first one identifies the transaction to which it belongs, and the second one the position that the task occupies within the tasks of its transaction, when they are ordered by increasing offsets. In this way, τ_{ij} is the *j*-th task of transaction Γ_i, with an offset of Φ_{ij} and a worst-case execution time of C_{ij}. In addition, each task is allowed to have jitter, that is, to have its activation time delayed by an arbitrary amount of time between 0 and the maximum jitter for that task, which we will call J_{ij}. This means that the activation of task τ_{ij} may occur at any time between $t_0+\Phi_{ij}$ and $t_0+\Phi_{ij}+J_{ij}$, where t_0 is the instant at which the external event arrived.

We allow deadlines to be larger than the period, and so at each time there may be several activations of the same task pending. We also allow both the offset Φ_{ij} and the jitter J_{ij} to be larger than the period of its transaction, T_i. For each task τ_{ij} we define its response time as the difference between its completion time and the instant at which the associated external event arrived. The worst-case response time will be called R_{ij}. Each task may have an associated global deadline, D_{ij}, which is also relative to the arrival of the external event.

The tasks can synchronize for using shared resources in a mutually exclusive way using a hard real-time synchronization protocol such as the priority ceiling protocol [9]. Under this assumption, the effects of lower priority tasks on a task under analysis τ_{ab} are bounded by an amount called the blocking term B_{ab}, calculated as the maximum of all the critical sections of lower priority tasks that have a priority ceiling higher than or equal to the priority of τ_{ab}.

2.2. Calculation of the worst-case response times

To calculate the worst-case global response time of a task τ_{ab}, we must build the worst-case scenario for its execution. To achieve this, we must create a critical instant that leads to the worst-case busy period. A task τ_{ab} busy period is an interval of time during which the CPU is busy processing task τ_{ab} or higher priority tasks. In tasks with offset we must take into account that the critical instant may not include the simultaneous activation of all higher priority tasks, as was the case when all tasks were independent. The existence of offsets makes it impossible for some sets of tasks to become active simultaneously. The analysis calculates the worst-case interference of a transaction Γ_i on the response time of a task τ_{ab} in the potentially critical instant that coincides with the most delayed activation of a task τ_{ik} belonging to the transaction:

$$\varphi_{ijk} = T_i - (\Phi_{ik}+J_{ik}-\Phi_{ij}) \bmod T_i \tag{1}$$

$$W_{ik}(\tau_{ab},t) = \sum_{\forall j \in hp_i(\tau_{ab})} \left(\left\lfloor \frac{J_{ij}+\varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t-\varphi_{ijk}}{T_i} \right\rceil \right) C_{ij} \tag{2}$$

where $hp_i(\tau_{ab})$ represents the set of tasks belonging to transaction Γ_i with priority higher than or equal to the priority of τ_{ab} and executing in the same processor as τ_{ab}. The main problem with this analysis technique is that we don't know which task τ_{ik} must be used to create the worst-case busy period. We obtain an upper bound of the interference of the tasks of a transaction Γ_i in a busy period of duration *w*, as the maximum of all possible interferences that could be caused by considering each of the tasks of Γ_i as the one originating the busy period:

$$W_i^*(\tau_{ab},w) = \max_{\forall k \in hp_i(\tau_{ab})} W_{ik}(\tau_{ab},w) \tag{3}$$

In order to introduce less pessimism, we will not use this function for the transaction to which the task under analysis belongs, but we will use the original transaction. Consequently, we must consider all the possible critical instants created with each of the task τ_{ac} in the set $hp_a(\tau_{ab})$, adding the own τ_{ab}. We calculate the completion time of each job *p* in the busy period, $w_{abc}(p)$, by means of:

$$\begin{aligned} w_{abc}(p) = {} & B_{ab}+(p-p_{0,abc}+1)C_{ab} + \\ & + W_{ac}(\tau_{ab},w_{abc}(p)) + \sum_{\forall i \neq a} W_i^*(\tau_{ab},w_{abc}(p)) \end{aligned} \tag{4}$$

Parameter $p_{0,abc}$ corresponds to the first activation that occurs at the critical instant:

$$p_{0,abc} = -\left\lfloor \frac{J_{ab}+\varphi_{abc}}{T_a} \right\rfloor + 1 \tag{5}$$

The length of the busy period is calculated as:

$$\begin{aligned} L_{abc} = {} & B_{ab}+\left(\left\lceil \frac{L_{abc}-\varphi_{abc}}{T_a} \right\rceil - p_{0,abc}+1 \right) C_{ab} + \\ & + W_{ac}(\tau_{ab},L_{abc}) + \sum_{\forall i \neq a} W_i^*(\tau_{ab},L_{abc}) \end{aligned} \tag{6}$$

and from it:

$$p_{L,abc} = \left\lceil \frac{L_{abc}-\varphi_{abc}}{T_a} \right\rceil \tag{7}$$

The global worst-case response time is obtained by subtracting the instant at which the associated event arrived, from the completion time :

$$R_{abc}(p) = w_{abc}(p) - \varphi_{abc} - (p-1)T_a + \Phi_{ab} \tag{8}$$

And then we need to take the worst of all the response times obtained:

$$R_{ab} = \max_{\forall c \in hp_a(\tau_{ab}) \cup b} \left[\max_{p=p_{0,abc}..p_{L,abc}} (R_{abc}(p)) \right] \quad (9)$$

This model can be applied as an approximation for the analysis of distributed systems, considering for each task τ_{ij} an equivalent offset and jitter term calculated with:

$$\begin{aligned} \Phi_{ij} &= R^{b}_{ij-1} \\ J_{ij} &= R_{ij-1} - R^{b}_{ij-1} + J_{clock} \end{aligned} \quad (10)$$

where R^{b}_{ij-1} is a lower bound for the best-case response time of the previous task τ_{ij-1} [6], and R_{ij-1} is an upper bound for the worst-case response time. With J_{clock} we model the effects due to a coarse clock resolution and/or non-perfect clock synchronization. Now, the main problem is that the response times are dependent on the task offsets, and the task offsets depend on the response times. For R^{b}_{ij-1}, we can use any lower bound to the best-case response time, including zero. The solution to this problem can be found in WCDO iterative method [7], based upon Tindell & Clark's holistic analysis [10]: starting from an initial value of response times of zero, we apply the analysis using the technique for static offsets with the equivalent offsets and jitter terms. In this way we obtain the response times of each task. Using these response times we re-calculate the equivalent jitter using (10), and with this new value we recalculate the response times. This calculation continues in an iterative way until we obtain the same result in two successive iterations, that is $R^{(n+1)}_{ij} = R^{(n)}_{ij} \quad \forall i, \forall j$.

3. Activation conflicts and priority schemes

Although the method based on dynamic offsets is a major improvement over previous methods that considered tasks as independent, it can be pessimistic in some situations, as is shown with the next example. Figure 1-a shows a periodic transaction Γ_i with 5 tasks of different priority levels executing in a single processor. The downward arrows indicate the occurrence of events and the upward arrows the offsets of the tasks, Φ_{ij}. Each box represents the execution of a task, its height being proportional to its assigned priority. Activations of tasks in different jobs are shown by different shading. The maximum jitter terms, J_{ij}, corresponding to the activation of each task are also represented with horizontal lines. We will study the contribution of tasks in Γ_i to the worst-case response time of another task τ_{ab} (whose priority is indicated by the dashed horizontal line),in the busy period starting at time t_c.

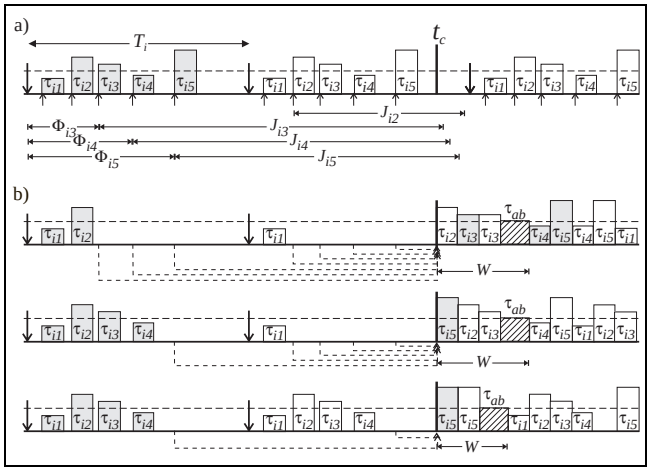

Figure 1. Possible execution scenarios of a transaction

Figure 1-b shows three possible execution scenarios, depending on the jitter chosen for each activation, which is indicated in the figure by the lower dashed lines. The first scenario represents the execution of the system when we choose the jitter in the same way we used to built the worst-case up to now, that is, delaying the activations corresponding to jobs before the critical instant until they coincide with it. If we ignore precedence relations, all task activations with priority higher than τ_{ab} will interfere in its execution so that the completion time would be, at least, $W=2C_{i2}+3C_{i3}+3C_{i5}+C_{ab}$. Nevertheless, the precedence relations oblige the execution of task τ_{i5} not to start until the execution of the preceding task τ_{i4} has finished. Moreover, given that task τ_{i4} has a lower priority than that assigned to τ_{ab}, τ_{i4} cannot start to execute until τ_{ab} has finished, as is shown in the first of the three situations in Figure 1-b. This means that the completion time of task τ_{ab} would in this case be equal to $W=C_{i2}+2C_{i3}+C_{ab}$. This situation represents a more realistic execution scenario, but it makes the worst-case analysis more difficult, since we cannot construct the critical instant in the same way as before. We would choose, for example, the activation time of τ_{i4} corresponding to the first job so that its execution would have finished before the critical instant and so, τ_i could be executed within the busy period. The second scenario in Figure 1-b shows an example of this case. The difficulty lies in that, by making τ_{i4} execute earlier to force τ_{i5} to execute within the busy period, the activation of τ_{i3} belonging to the same job also has to finish before the critical instant and, therefore, cannot interfere in the execution of task τ_{ab}. This originates a conflict when deciding which of the two situations we must consider for the worst-case, given that one or other may sometimes contribute more to the response time, depending on the execution time of each task. Exactly the same problem occurs in the second job, since the execution of its tasks τ_{i2} and τ_{i3} is incompatible with the execution of its task τ_{i5}, as shown in the last two scenarios. We will call these incompatible situations conflicts. Now, let us define more formally the concept of conflict.

Definition 5-1. Two tasks are in conflict when the execution of one is incompatible with the execution of the other, within the same busy period. Let τ_{ij} and τ_{ik} be two tasks with higher or equal priority than that assigned to another task τ_{ab}, and suppose that τ_{ij} precedes τ_{ik}, $j<k$. If,

in the transaction Γ_i there is a intermediate task τ_{il} ($j<l<k$) in the same processor and with a lower priority than τ_{ab}, then in a busy period of τ_{ab}, tasks τ_{ij} and τ_{ik} activated in the same job cannot interfere τ_{ab} simultaneously. In this case, the activation of tasks τ_{ij} and τ_{ik} is said to be in conflict.

We identify each task activation depending on the instant of the arrival of the event that triggered its execution. We assign consecutive positive numbers to the events arriving after the critical instant, assigning p'=1 to the tasks activated in the transaction Γ_i triggered by an event which arrived in the interval $(0,T_i]$, p'=2 to an event arriving in $(T_i,2T_i]$, etc. In the same way, the events occurring before the critical instant are identified with consecutive numbers p'≤0; p'=0 corresponding to the tasks of the transaction triggered in the interval $(-T_i,0]$, p'=-1 to the interval $(-2T_i,-T_i]$, etc. In this way, we identify with the same index activations that have precedence relations and correspond to the same job. In the first of the two jobs of Figure 1, with index p'=-1, there is a conflict between the activations of the tasks τ_{i3} and τ_{i5}. In the second (p'=0) there is a conflict between the activation of tasks τ_{i2} and τ_{i3} and the activation of task τ_{i5}. Note that the execution of the tasks τ_{i2} and τ_{i3} is compatible in the same busy period.

To locate the activation conflicts we will identify the pending activations of the tasks of a transaction Γ_i in a specific busy period. In Figure 2 three activation scenarios are described for a task τ_{ij}, which will help with the identification, and show a critical instant t_c delayed an amount ϕ with respect to the arrival of the events triggering the transaction Γ_i. Scenario 1 corresponds to the case $\phi \geq \Phi_{ij}$ while Scenario 2 shows the case $\phi < \Phi_{ij}$; finally, Scenario 3 corresponds to a task with offset greater than the period, $\Phi_{ij} > T_i$, and $\phi < \Phi_{ij}$ (if ϕ were greater than Φ_{ij} we would refer to the next event, with a new $\phi' < \Phi_{ij}$). Once again we represent the arrival of events with downward arrows and the activation of tasks with upward arrows. The continuous horizontal lines represent the offsets, and the dashed lines represent the jitter delay of the activations. We calculate the number n'_{ij} of events occurring before the critical instant and that are awaiting the execution of task τ_{ij} at the critical instant. In the figure, the events arriving at the instants t_0, t_1 and t_2 are found in this situation so, in the three situations $n'_{ij}=3$.

For the calculation of n'_{ij} we define the magnitude φ' as the interval between the critical instant and the activation of τ_{ij} corresponding to the arrival of the first event after the critical instant (identified with p'=1). As can be seen, this value is equal to:

$$\varphi' = T_i - \phi + \Phi_{ij} \tag{11}$$

Note that the value φ' can be greater than the period, for $\phi < \Phi_{ij}$, even several times, if the offset is sufficiently big.

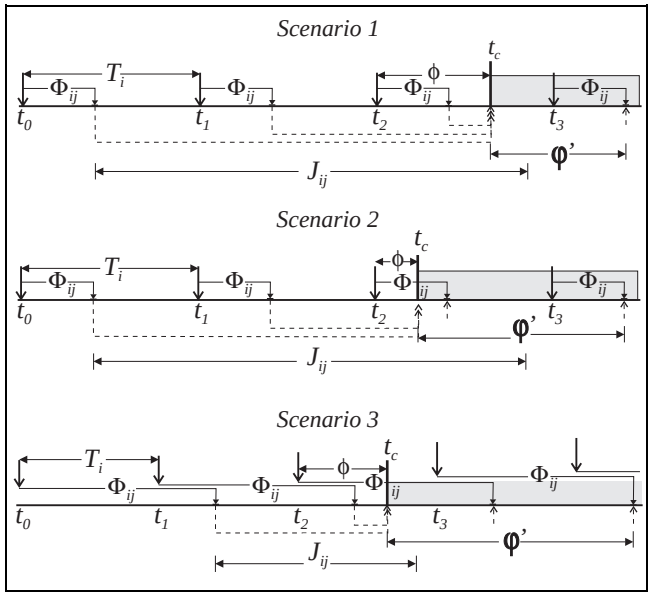

Figure 2. Execution scenarios

Under the conditions when the potential critical instant is created, t_0 corresponds to the instant of arrival of the first event with the execution of the task τ_{ij} still to be started at the beginning of the busy period. Therefore, this first job must simultaneously verify the following two inequalities:

$$\begin{aligned} t_0 + \Phi_{ij} + J_{ij} &\geq t_c \\ t_0 - T_i + \Phi_{ij} + J_{ij} &< t_c \end{aligned} \tag{12}$$

Focusing on Figure 2, we can see that the critical instant t_c can be expressed as:

$$t_c = t_0 + n'_{ij}\, T_i + \Phi_{ij} - \varphi' \tag{13}$$

which substituting into the two previous expressions:

$$\begin{aligned} t_0 + \Phi_{ij} + J_{ij} &\geq t_0 + n'_{ij}\, T_i + \Phi_{ij} - \varphi' \\ t_0 - T_i + \Phi_{ij} + J_{ij} &< t_0 + n'_{ij}\, T_i + \Phi_{ij} - \varphi' \end{aligned} \tag{14}$$

leads to:

$$n'_{ij} \leq \frac{J_{ij}+\varphi'}{T_i} \qquad \textit{and} \qquad n'_{ij} > \frac{J_{ij}+\varphi'}{T_i} - 1 \tag{15}$$

Given that n'_{ij} is a integer number, the only solution of these two inequalities is:

$$n'_{ij} = \left\lfloor \frac{J_{ij}+\varphi'}{T_i} \right\rfloor \tag{16}$$

If we apply this result at the critical instant t_c created with the task τ_{ik}, in which the activation delay is equal to $\phi = (\Phi_{ik}+J_{ik})\ mod\ T_i$, we obtain the interval φ'_{ijk} between this critical instant and the activation of the task τ_{ij} corresponding to the first event occurring after t_c:

$$\varphi'_{ijk} = T_i - (\Phi_{ik} + J_{ik}) \bmod T_i + \Phi_{ij} \tag{17}$$

which, substituted into (16), gives the value of the number n'_{ijk} of events pending execution of the task τ_{ij} at the beginning of the busy period:

$$n'_{ijk} = \left\lfloor \frac{J_{ij}+\varphi'_{ijk}}{T_i} \right\rfloor \quad (18)$$

Following the previously defined numbering scheme, the last of these pending events has the index $p'=0$; therefore, the first of them is identified with the value $p'_{0,ijk}$ calculated:

$$p'_{0,ijk} = -n'_{ijk} + 1 = -\left\lfloor \frac{J_{ij}+\varphi'_{ijk}}{T_i} \right\rfloor + 1 \quad (19)$$

In the calculation of the maximum interference due to a transition Γ_i we must determine and resolve the possible conflicts of activation in the busy period that starts at the critical instant under consideration. Resolving a conflict means choosing the execution of those tasks that have the greatest interference with the task under analysis, τ_{ab}. It is important to highlight that the activations corresponding to instances with index $p'\geq 1$ are produced after the critical instant and, therefore, the task activation phase cannot be brought forward in order to resolve possible conflicts. In this case, the tasks must execute according to the order of precedence, starting with the first task of the transaction in the same processor. In the example in Figure 1, it can be seen that the first task of the transaction, τ_{i1}, has lower priority than τ_{ab} and so, no task corresponding to jobs posterior to the instant t_c can interfere in the busy period.

To determine and resolve the activation conflicts, we must characterize the priority scheme of each transaction. From the point of view of the analysis of a task τ_{ab} we will classify the tasks of a transaction Γ_i in different sections, which we will call *H* sections. An *H* section is composed of a set of contiguous tasks (in the processor where τ_{ab} executes) with a priority higher than or equal to that assigned to τ_{ab}.

Definition 5-2. Two tasks τ_{ij} and τ_{ik} of a transaction Γ_i belong to the same *H* section, for the analysis of a task τ_{ab}, if both execute in the same processor as τ_{ab} with priority higher than or equal to that assigned to τ_{ab} and there is no intermediate task of the same transaction executing in the same processor with priority lower than that of τ_{ab}. We will identify with $H_{ij}(\tau_{ab})$ the section to which task τ_{ij} belongs, and that is made up of the set of tasks of the transaction Γ_i which verify:

$$H_{ij}(\tau_{ab}) = \left\{ l \in \Gamma_i \mid \left(\nexists \left[j \leq x \leq l \vee l \leq x \leq j \right] \mid proc(\tau_{ix})=proc(\tau_{ab}) \wedge prio(\tau_{ix})<prio(\tau_{ab})\right) \right\} \quad (20)$$

Tasks τ_{ij} and τ_{ik} will belong to the same section if $H_{ij}(\tau_{ab})=H_{ik}(\tau_{ab})$ is fulfilled. In the example in Figure 1, from the point of view of the execution of the task τ_{ab}, the transaction Γ_i is made up of two different *H* sections, which are: the section $H_{i2}(\tau_{ab})=H_{i3}(\tau_{ab})=\{\tau_{i2},\tau_{i3}\}$ and the section $H_{i5}(\tau_{ab})=\{\tau_{i5}\}$. Note that, according to definitions 1 and 2, two tasks will be in conflict if and only if they execute in the same processor and belong to different *H* sections. If in a specific busy period there is a job with various tasks of different conflicting sections, we must choose for the worst-case those tasks belonging to a single *H* section whose sum of execution times is the greatest. In the example in Figure 1 we have two jobs with tasks that can be delayed until the instant t_c. The first job has conflicting activations of tasks τ_{i3} and τ_{i5}, belonging to different sections, therefore, we will choose for the worst-case the one with greatest execution time, equal to $Max(C_{i3},C_{i5})$. The second job has the activation of τ_{i2} and τ_{i3} in conflict with the activation of τ_{i5}, since the first two belong to the same *H* section, different to that of τ_{i5}, in such a way that for the worst-case we choose $Max(C_{i2}+C_{i3},C_{i5})$. The maximum total contribution to the busy period will be the one due to both jobs, giving a bound:

$$W = Max(C_{i3},C_{i5}) + Max(C_{i2}+C_{i3},C_{i5}) + C_{ab} \quad (21)$$

We can check that this result is the same as obtained from calculating the maximum of the three execution schemes considered in Figure 1-b. Remember that if we apply the techniques without considering the precedence relationships, we obtain at least a value $W=2C_{i2}+3C_{i3}+3C_{i5}+C_{ab}$.

4. Worst-case interference

We will deduce the expression for the maximum interference due to the tasks of transaction Γ_i in the execution of another task τ_{ab}, by studying all the possible critical instants created with tasks of Γ_i with higher or equal priority than τ_{ab}. We will focus on the study of the critical instant created with a task τ_{ik}. For this purpose, we distinguish between the jobs initiated before or after the critical instant. All the jobs posterior to the critical instant, with indices $p'\geq 1$, must start execution in order of precedence beginning with the first task; therefore, only the tasks belonging to the first *H* section can contribute to the busy period. We identify these as:

$$MP_i(\tau_{ab}) = \left\{ l \in \Gamma_i \mid \left(\nexists\, x<l \mid proc(\tau_{ix})=proc(\tau_{ab}) \wedge prio(\tau_{ix})<prio(\tau_{ab})\right)\right\} \quad (22)$$

and given that the activation corresponding to $p'=1$ is produced at the instant φ'_{ijk} calculated with (17), the expression for this interference is:

$$W_{ik}(\tau_{ab},t)\Big|_{p'>0} = \sum_{\forall j \in MP_i(\tau_{ab})} \left\lceil \frac{t-\varphi'_{ijk}}{T_i} \right\rceil_0 C_{ij} \quad (23)$$

Since φ'_{ijk} can be greater than T_i and the contribution to the response time cannot be negative, we must consider the maximum with 0, and so $\lceil x \rceil_0 = max(\lceil x \rceil,0)$.

For the contribution to the worst-case of jobs previous to the critical instant created with τ_{ik}, we must find and resolve possible execution conflicts between tasks of Γ_i. To do this, we will construct a so-called conflict table in the

busy period of width t for the task τ_{ab}. Each row represents a vector of the tasks belonging to transaction Γ_i, while each column represents the different jobs of the transaction (with values $p' \leq 0$). Each cell (j,p') can take one of two different values: 0 if the activation p' of task τ_{ij} has not been produced within the interval $[0,t)$, or C_{ij} —the worst-case execution time— if activation p' of τ_{ij} has been produced within the busy period of width t. Given that activation $p'=1$ is originated at instant φ'_{ijk}, the activation numbered p' will be released at instant $\varphi'_{ijk} - (p'-1)T_i$.

We must construct a table of, at most, N columns and n'_{iNk} rows, N being the index of the last task in the set $hp_i(\tau_{ab})$ and n'_{iNk} the number of pending activations of this last task. Next, we show the algorithm for creating the conflict table for transaction Γ_i, corresponding to the critical instant started with task τ_{ik}, for the analysis of a task τ_{ab}.

During the elaboration of the table we can reduce the number of conflicts since when we study the critical instant created with the task τ_{ik} we are implicitly eliminating the possibility of some conflicts. Those tasks preceded by τ_{ik} entering in conflict with it can not execute since we have already chosen the execution of τ_{ik} at least for the event with which the critical instant is created, with an index equal to $p'_{0,ikk}$, given by (19). This means that in those jobs with index greater than or equal $p'_{0,ikk}$, we can eliminate the activations of tasks preceded by τ_{ik} not belonging to the same H section, according to the following algorithm:

```
Procedure BUILD_CONFLICTS_TABLE(τab, t, Γi,τik, out Table)is
begin
  Initialize_Table;
  For all τij ∈ hpi(τab) loop
    For p' in p'0,ijk .. 0 loop
        if t ≥ φ'ijk - (p'-1)Ti then Table(j,p') := Cij;
        else Table(j,p') := 0;
        if p'≥p'0,ikk and j>k and Hij(τab)≠Hik(τab) then Table(j,p'):=0;
    end loop;
  end loop;
end BUILD_CONFLICTS_TABLE;
```

To resolve the activation conflicts we must go through the table by rows (jobs), obtaining for each one the maximum execution time required by the processor for an H section. The execution time required by each H section in the job p' is calculated by adding, in the row p', the columns corresponding to tasks belonging to this section. The total interference is obtained by adding the values obtained for each row, as in the algorithm shown below.

In practice, building the conflicts table is not necessary because it is possible to integrate the latter two algorithms into a single one that obtains and then resolves each of the conflicts in each pending activation [8].

Considering the resolution of conflicts in jobs before the

```
Function RESOLVE_CONFLICTS(τab, t, Γi,τik) return time is
begin
  BUILD_CONFLICTS_TABLE(τab, t, Γi,τik, Table);
  Total:=0;
  For p' in p'0,iNk .. 0 loop
    max_section:=0; sum:=0;
    For all τij in Γi loop
        if prio(τij)<prio(τab) and proc(τij)=proc(τab) then sum:=0;
        else sum:=sum+Table(j,p');
        if sum>max_section then max_section:=sum;
    end loop;
    Total:=Total+max_section;
  end loop;
  return Total;
end RESOLVE_CONFLICTS;
```

critical instant with $p' \leq 0$ and equation (23) for $p'>0$, we obtain the interference due to the tasks of transaction Γ_i in the busy period of width t starts in the critical instant created with τ_{ik}, as

$$W_{ik}(\tau_{ab},t) = RESOLVE\_CONFLICTS(\tau_{ab},t,\Gamma_i,\tau_{ik}) + \left\lceil \frac{t-\varphi'_{ijk}}{T_i} \right\rceil_0 C_{ij}$$

Starting from this equation we can derive the function $W^*_i(\tau_{ab},t)$, which obtains an upper bound of the maximum interference due to tasks of the transaction Γ_i

$$W_i^*(\tau_{ab},t) = \max_{\forall k \in hp_i(\tau_{ab})} W_{ik}(\tau_{ab},t) \qquad (25)$$

Next, we will illustrate this technique applying it to the example shown in Figure 1. We will analyze the possible worst-case situations constructed with each of the tasks with priority higher than τ_{ab}. In Figures 3 to 5 each of the three constructed scenarios is shown, according to the critical instant that has been created with τ_{i2}, τ_{i3}, or τ_{i5}.

Figure 3 shows the case of the critical instant created when we activate task τ_{i2} undergoing its maximum possible delay. As can be observed, at instant t_c the execution of tasks τ_{i2}, τ_{i3}, and τ_{i5} corresponding to the job numbered $p'=0$ is pending, and so we create the table of conflicts shown at the bottom of the figure. Note that we can eliminate the cell corresponding to the task τ_{i5} from the table, given that the critical instant created with τ_{i2} would be incompatible with its execution, as is indicated in the algorithm for creating the conflict table. In fact, there is no activation conflict since the only possible scenario is the one

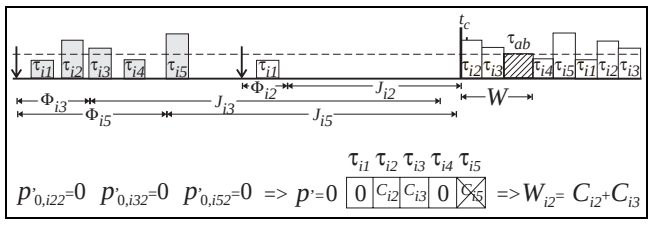

Figure 3. Critical instant created with task τ_{i2}

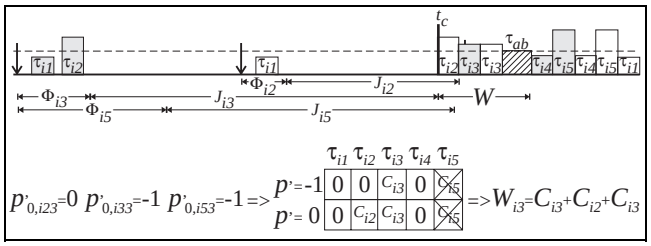

Figure 4. Critical instant created with task τ_{i3}

shown in the figure. The resolution of conflicts is shown on the right of the table: given that tasks τ_{i2} and τ_{i3} belong to the same *H* section, the result in the only job (p'=0) of the table is equal to $C_{i2}+C_{i3}$. The set of tasks MP_i is empty, given that the first task of the transaction executes in the same processor and has a priority lower than that assigned to τ_{ab}, and therefore, the contribution of posterior jobs to the critical instant, with indices p'>0, will be null. The possible contribution to the worst case of τ_{ab} in the busy period studied will be $W_{i2}=C_{i2}+C_{i3}$.

Figure 4 shows the critical instant constructed with τ_{i3}. In this case, the job numbered p'=0 has the execution of tasks τ_{i2}, τ_{i3} and τ_{i5} pending, but the executions of τ_{i3} and τ_{i5} corresponding to the previous job (p'=-1) are also pending. The table will therefore have two rows, as is shown in the figure, in which the executions incompatible with the critical instant are eliminated. For each of these two jobs, the maximum section is calculated and they are added. The estimated bound in this case for the contribution to the busy period is equal to $W_{i3}=C_{i3}+C_{i2}+C_{i3}$.

Finally, Figure 5 shows the two possible scenarios when the critical instant is created with the task τ_{i5}. In the instant t_c, one job of tasks τ_{i2} and τ_{i3} and two jobs of task τ_{i5} are pending. Note that in this case, by applying the reduction as in other cases, we do not eliminate any activation from the table. Furthermore, now we have a conflict in the table (in the previous two cases there were no effective conflicts). The job p'=-1 contributes with C_{i5}, but in the job p'=0 we have activations in two different sections that can contribute to the busy period. The resolution of this conflict obtains a contribution, for this job, equal to the maximum of the contribution of the two sections, *Max* ($C_{i2}+C_{i3}$, C_{i5}). The total contribution at the critical instant created with the task

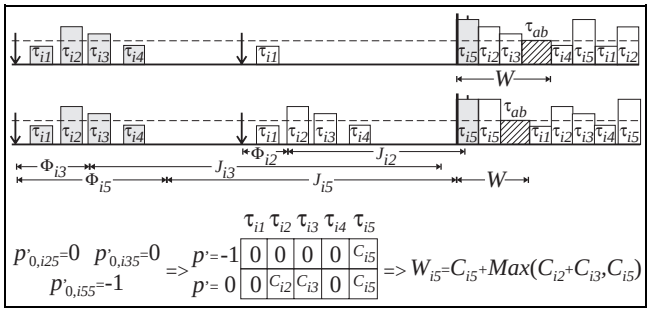

Figure 5. Critical instant created with task τ_{i5}

τ_{i5} is $W_{i5} = C_{i5} + Max$ ($C_{i2}+C_{i3}$, C_{i5}).

The maximum contribution of tasks of transaction Γ_i to the worst-case response of task τ_{ab} can therefore be bound with the expression:

$$W_i^* = Max(W_{i2},W_{i3},W_{i5}) = Max\left(C_{i2}+2C_{i3}, C_{i5}+Max(C_{i2}+C_{i3}, C_{i5})\right)$$

It can be seen that this expression is equivalent to the one obtained in equation (21), obtained by inspection of the example in Figure 1.

In real-time systems with precedence relationships, in which one task is activated when the previous one finalizes, it is possible to reduce the pessimism in the analysis, also reducing the number of cases to analyze, if we bear in mind the following lemma:

Lemma 5-1. Let τ_{ij} and τ_{ij+1} be two consecutive tasks in the same transaction Γ_i, such that task τ_{ij+1} is activated when the execution of task τ_{ij} finalizes. If both tasks are located in the same processor, there cannot be a busy period of level *pr* = minimum(prio(τ_{ij}) , prio(τ_{ij+1})) or lower, whose start time coincides with the activation of task τ_{ij+1}.

Proof. Suppose there were a busy period of level *pr* or lower initiated at an instant *t* at which task τ_{ij+1} is activated. Given that task τ_{ij+1} is activated immediately after task τ_{ij}, just at the instant *t* the execution of τ_{ij} will have finished and, given that they execute in the same processor, this means that the start of the busy period cannot be the instant *t*, but it must go back, at least, to the instant of the activation of task τ_{ij}. Note that this reasoning would not be valid if task τ_{ij+1} had a priority higher than that of task τ_{ij} and we were studying an intermediate level busy period.□

This lemma allows us to reduce the number of cases to be considered for the task τ_{ab}. If in a transaction Γ_i, two consecutive tasks execute in the same processor as τ_{ab} and they are assigned higher or equal priorities than τ_{ab}, then it is sufficient to analyze the busy period that starts in the possible critical instant created with the first of the tasks. Extending this reasoning, it is only necessary to analyze the possible critical instants created with the tasks in the set:

$$XP_i(\tau_{ab}) = \left\{ l \in \Gamma_i \;\middle|\; \begin{array}{l} proc(\tau_{il}) = proc(\tau_{ab}) \wedge prio(\tau_{il}) \geq prio(\tau_{ab}) \wedge \\ proc(\tau_{il-1}) \neq proc(\tau_{ab}) \vee prio(\tau_{il-1}) < prio(\tau_{ab}) \end{array} \right\}$$

that is, the set of tasks of the transition Γ_i executing in the same processor as task τ_{ab} with higher or equal priority than that assigned to τ_{ab} and whose predecessor, if there is one, is not found in the same conditions. The expression to obtain an upper bound of the maximum interference due to tasks of transaction Γ_i is as follows;

$$W_i^*(\tau_{ab},t) = \max_{\forall k \in XP_i(\tau_{ab})} W_{ik}(\tau_{ab},t) \qquad (28)$$

where $W_{ik}(\tau_{ab},t)$ corresponds to equation (24). Given that the set $XP_i(\tau_{ab})$ can contain fewer tasks that the set $hp_i(\tau_{ab})$ defined previously, we eliminate the analysis of non-possible busy periods, thus, reducing the pessimism

introduced in the analysis and the number of cases to be analyzed, making the algorithm faster.

5. Application to the transaction under analysis

We are now going to apply the considerations about the precedence relations and priority schemes to the transaction to which the task under analysis τ_{ab} belongs. The methodology of analysis estimates upper bounds on the response times of a task by studying the possible critical instants created with each of the tasks of a transaction Γ_a with a priority higher than or equal to that of the task under analysis, including the task itself. In this section, we derive the expression for the contribution of tasks of Γ_a to the worst case of a task τ_{ab}, in the busy period starting in the instant of maximum jitter of the activation of a task τ_{ac}.

Figure 6-a shows an example of a transaction Γ_a with four tasks executing in a processor (supposing that the rest of the tasks are executing in others processors). Once again, the downward arrows indicate the occurrence of events and the upward arrows the offsets of the tasks. Each box represents the execution of a task, its height being proportional to its assigned priority. Activations of tasks in different jobs are represented by different shadings, and maximum activation delays by continuous lines. We will consider the critical instant created through task τ_{a5} for the analysis of task τ_{a3}. In these conditions, the transaction has two *H* sections: one made up of a task τ_{a1} and another made up of the tasks τ_{a3} and τ_{a5}. As can be observed, in that critical instant there are three jobs that have the execution of some of their tasks pending, so we must take into account the activation conflicts between tasks of different *H* sections. The lower part shows the three possible execution scenarios depending on the tasks chosen for execution. The first scenario represents the execution when the two pending activations of task τ_{a1} are delayed until the critical instant, with indices p'=-1 and p'=0. The second represents the execution when only the one corresponding to p'=0 is delayed, and third when neither is delayed (or they are not delayed enough so as to occur after the critical instant).

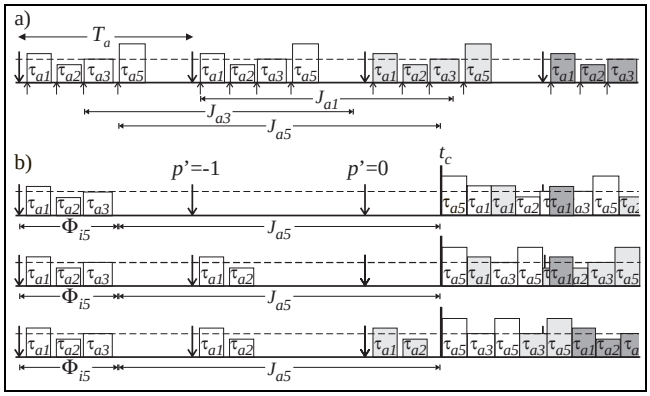

Figure 6. Critical instant created with task τ_{a5}

Although the three execution scenarios start with a possible critical instant created with τ_{a5}, they are not all valid for the analysis of the task τ_{a3}. We will focus on the first scenario created, in which the pending activations of task τ_{a1} are delayed so as to execute within the busy period. Given that tasks τ_{a1} and τ_{a3} belong to different *H* sections, the corresponding busy period does not contain any execution of the task τ_{a3} and, thus, is not useful for the analysis. The other two scenarios, however, do contain executions of task τ_{a3} and must be analyzed.

This means that we can reduce the number of possible cases to be considered in the analysis. As was deduced in the previous section, when considering the critical instant created with a task τ_{ac} we must bear in mind that the execution of this task τ_{ac} has been forced to execute inside the associated busy period, at least for the job with which the critical instant was created. For the analysis of task τ_{ab} we are also forcing another situation: that the activation analyzed is within the constructed busy period. Once again, focusing on the example in Figure 6, the busy period represented in the third scenario contains the execution of task τ_{a3} corresponding to the jobs p'=-1 and p'=0. However, the second scenario contains only the execution of τ_{a3} belonging to the job p'=-1, since for the job p'=0 the execution of task τ_{a1} was chosen for the busy period. This means that in the analysis of the job p'=-1 we must consider the two scenarios, but in the analysis corresponding to the job p'=0 it is sufficient to analyze the last scenario, given that the others are incompatible with the execution of τ_{a3} in the busy period.

Therefore, we can extend the rules applicable for the reduction of the conflict table in the analysis of task τ_{ab} corresponding to the job p'_{ab} in the busy period created with task τ_{ac}. On the one hand, the reduction rule mentioned above, due to the creation of the critical instant. On the other hand, the conflicts in the job p'_{ab} and the previous ones will have had to be resolved in a way compatible with the execution, within the busy period considered, of the activation p'_{ab} of τ_{ab}. In the example, if for the jobs p'=-1 and p'=0, task τ_{a1} has executed inside the busy period, it is not necessary to analyze the execution of τ_{a3} corresponding to the job p'_{a3} =0. These reduction rules are:

1st Reduction rule (due to the creation of a critical instant). In the conflict table, the activations corresponding to $p'_{0,acc}$ and posterior jobs entering in conflict with task τ_{ac} can be eliminated. That is, we can eliminate the cells (j,p') verifying the condition.

$$p' \geq p'_{0,acc} \quad \wedge \quad j>c \quad \wedge \quad H_{aj}(\tau_{ab}) \neq H_{ac}(\tau_{ab}) \qquad (29)$$

2nd Reduction rule (due to the execution of the job p'_{ab} of τ_{ab} in the busy period). We can eliminate the conflicts in

jobs p'_{ab} and anterior jobs, incompatible with the execution of τ_{ab}. That is:

$$p' \leq p'_{ab} \ \wedge \ j<b \ \wedge \ H_{aj}(\tau_{ab}) \neq H_{ab}(\tau_{ab}) \qquad (30)$$

Furthermore, we can consider another reduction rule in the analysis. The tasks preceded by τ_{ab} can not interfere in the execution of a job p'_{ab} if they correspond to jobs posterior to the analysis. That is take into account in the next reduction rule:

3rd Reduction rule (due to the analysis of the job p'_{ab} of τ_{ab}). The cells corresponding to the tasks preceded by τ_{ab} in the same or posterior jobs can be eliminated from the conflict table. The activations posterior to the task τ_{ab} itself do not have to be considered either. Therefore, we can eliminate the cells (j,p') which fulfil:

$$(p' \geq p'_{ab} \ \wedge \ j>b) \ \vee \ (p' > p'_{ab} \ \wedge \ j=b) \qquad (31)$$

With all these considerations, the table of conflicts in the transaction Γ_a for the analysis of the job p'_{ab} of τ_{ab} can be created according to an algorithm *BUILD_CONFLICTS_TABLE_IN_*Γ_a similar to the one used to create a conflict table in the previous section, but adding the new reduction rules. The process of resolution of the conflicts is similar to the function *RESOLVE_CONFLICTS* described in the previous section, but using this last algorithm to create the table.

Through the resolution of the activation conflicts we obtain the interference due to jobs activated before the critical instant. Another aspect we still have to consider is the preemption due to tasks activated in jobs posterior to the critical instant. To obtain the expression of this interference we must take two effects into account due to the precedence relationships in the tasks of transaction Γ_a and which, in part, we have already taken into account to reduce the conflict table.

- Jobs posterior to the critical instant can only interfere in the busy period with tasks belonging to the first section H, included in the set MP_a.
- In the analysis of the activation p'_{ab} of a task τ_{ab} there cannot be interference due to tasks preceded by it, activated in the same job p'_{ab} or posterior.

The activation of a task τ_{aj} corresponding to the first job after the critical instant (with index p'=1), is by definition produced in the instant φ'_{ajc}. Therefore, a task τ_{aj} of the set MP_a will be activated periodically every T_a starting from instant φ'_{ajc}.

We differentiate the contribution of tasks belonging to the set MP_a depending on whether they precede or are preceded by the task under analysis. The tasks preceding τ_{ab} have no restriction in terms of their possible preemptions, so that their contribution to the busy period of width t will be given by the expression:

$$\sum_{\substack{\forall j \in MP_a(\tau_{ab}) \\ j<b}} \left\lceil \frac{t-\varphi'_{ajc}}{T_a} \right\rceil_0 C_{aj} \qquad (32)$$

The tasks preceded by τ_{ab} do have a limit in terms of their possible preemptions, since they can only preempt the execution of their activation p'_{ab} if they correspond to jobs before the job p'_{ab}; so their contribution will be:

$$\sum_{\substack{\forall j \in MP_a(\tau_{ab}) \\ j>b}} \min \left(p'_{ab}-1 \ , \left\lceil \frac{t-\varphi'_{ajc}}{T_a} \right\rceil_0 \right) C_{aj} \qquad (33)$$

Finally, we consider the contribution to the busy period of the task under analysis itself. In the conflict table, the activations of jobs before the critical instant were taken into account and t, therefore, we only have to consider those jobs activated after the critical instant (with indices p'>0) until the activation analyzed p'_{ab}, that is

$$p'_{ab} \ C_{ab} \qquad (34)$$

Note that these last two expressions only make sense for the analysis of the activations of τ_{ab} corresponding to jobs posterior to the critical instant and so, for negative values of p'_{ab} they must be equal to 0. In summary, the contribution to the worst case of tasks activated by events occurring after the critical instant, including task τ_{ab} itself, can be expressed as follows:

$$W_{ac}(\tau_{ab},t)\big|_{p'_{ab}>0} = \sum_{\substack{\forall j \in MP_a(\tau_{ab}) \\ j<b}} \left\lceil \frac{t-\varphi'_{ajc}}{T_a} \right\rceil_0 C_{aj} + \\ + \max\left\{0, p'_{ab} C_{ab} + \sum_{\substack{\forall j \in MP_a(\tau_{ab}) \\ j>b}} \min\left(p'_{ab}-1, \left\lceil \frac{t-\varphi'_{ajc}}{T_a} \right\rceil_0 \right) C_{aj} \right\} \qquad (35)$$

Thus, the total contribution of tasks of transaction Γ_a to the busy period for the activation p'_{ab} of task τ_{ab} is calculated as:

$$W_{ac}(\tau_{ab},p'_{ab},t) = RESOLVE\_CONFLICTS\_IN\_\Gamma_a(\tau_{ab},p'_{ab},t,\Gamma_a,\tau_{ik}) \\ + W_{ac}(\tau_{ab},p'_{ab},t)\big|_{p'_{ab}>0}$$

This expression completes the new analysis technique, in which we consider the precedence relationships and the priority schemes of the transactions. In this technique, the completion time $w_{abc}(p'_{ab})$ of an activation p'_{ab} is calculated according to the expression:

$$w_{abc}(p'_{ab}) = B_{ab} + W_{ac}\left(\tau_{ab},p'_{ab},w_{abc}(p'_{ab})\right) + \sum_{\forall i \neq a} W_i^*\left(\tau_{ab},w_{abc}(p'_{ab})\right)$$

where $W_i^*(\tau_{ab},w_{abc}(p'_{ab}))$ corresponds to equation (28). Using this result, the worst-case global response time, $R_{abc}(p'_{ab})$, is determined as:

$$R_{abc}(p'_{ab}) = w_{abc}(p'_{ab}) - \varphi'_{abc} - (p'_{ab}-1)T_a + \Phi_{ab} \qquad (38)$$

We must extend this analysis to all the activations occurring in the busy period L_{abc}, calculated using the equation:

$$L_{abc} = B_{ab} + W_{ac}(\tau_{ab}, L_{abc}) + \sum_{\forall i \neq a} W_i^*(\tau_{ab}, L_{abc}) \quad (39)$$

where $W_{ac}(\tau_{ab},L_{abc})$ is given by (24). We must analyze the activations with indices between $p'_{0,abc}$ and $p'_{L,abc}$, obtain from the busy period as:

$$p'_{L,abc} = \left\lfloor \frac{L_{abc} - \varphi'_{abc}}{T_a} \right\rfloor_0 \quad (40)$$

Nevertheless, if task τ_{ab} does not belong to the first section *H*, a busy period can not contain executions of jobs originated after the critical instant, with indices $p'>0$, so is sufficient to analyze until:

$$p'_{L,abc} = 0 \qquad if \quad \tau_{ab} \notin MP_a(\tau_{ab}) \quad (41)$$

The response time will correspond to the greatest of all of them, that is:

$$R_{ab} = \max_{\forall c \in XP_a(\tau_{ab})} \left[\max_{p=p'_{0,abc}..p'_{L,abc}} \left(w_{abc}(p'_{ab}) - \varphi'_{abc} - (p'_{ab}-1)T_a + \Phi_{ab} \right) \right] \quad (42)$$

We will see how this technique is applied in the example shown in Figure 6, for the analysis of task τ_{a3} in the potential critical instant created with task τ_{a5}. Figure 7-a shows the table of conflicts corresponding to the jobs before the critical instant. As can be seen, there are two pending activations of task τ_{a3} corresponding to the indices $p'=-1$ and $p'=0$. Figure 7-b shows the tables of conflicts generated for the analysis of each of these two activations, resulting from applying the reduction rules seen previously on the original table of conflicts. In the same way, the lower part of each table shows the result obtained after the resolution of these conflicts.

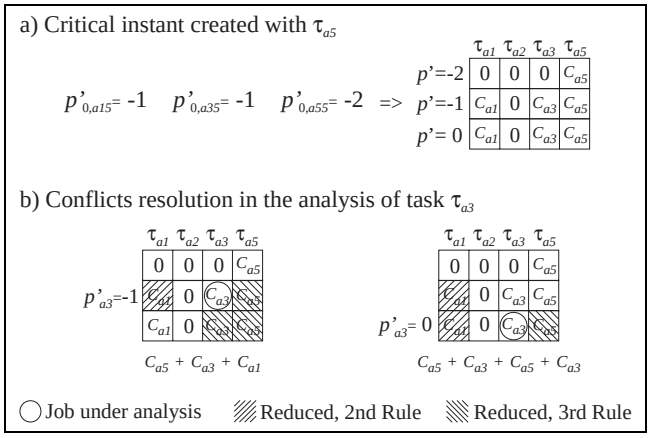

Figure 7. Analysis applied to task τ_{a3}

If we apply the criteria defined for the number of activations to be analyzed, we could see that we only have to carry out the analysis for values between $p'_{0,a35}$=-1 to $p'_{L,a35}$=0 since task τ_{a3} does not belong to the set $MP_a(\tau_a)$ made up only of task τ_{a1}.

The contribution to the worst case of τ_{a3}, in its first analyzed activation, with p'_{ab}=-1 is:

$$W_{a5}(\tau_{a3}, -1, t) = C_{a5} + C_{a3} + C_{a1} + \left\lfloor \frac{t - \varphi'_{a15}}{T_a} \right\rfloor_0 C_{a1} \quad (43)$$

which we will use to calculate the completion time $w_{a35}(-1)$ and, as a result, the corresponding global response time

$$R_{a35}(-1) = w_{a35}(-1) - \varphi'_{a35} + 2T_a + \Phi_{a3} \quad (44)$$

For the second activation, with p'_{ab}=0, the contribution to the worst case is calculated as:

$$W_{a5}(\tau_{a3}, 0, t) = C_{a5} + C_{a3} + C_{a5} + C_{a3} + \left\lfloor \frac{t - \varphi'_{a15}}{T_a} \right\rfloor_0 C_{a1} \quad (45)$$

from which we obtain $w_{a35}(0)$ and, from it, the response time as:

$$R_{a35}(0) = w_{a35}(0) - \varphi'_{a35} + T_a + \Phi_{a3} \quad (46)$$

In this way, the response time of the task τ_{a3}, in any busy period starting with a critical instant created by a task τ_{a5}, is bounded by the value

$$R_{a35} = Max(R_{a35}(-1), R_{a35}(0)) \quad (47)$$

Integrating this new formulation in the iterative algorithm for dynamic offsets, we will obtain better estimations of the worst-case response times in systems with precedence relations in the activation of their tasks. We will call this new algorithm WCDOPS (Worst-Case Dynamic Offsets with Priority Schemes). The complexity of this algorithm is approximately the same as for the previous algorithm (WCDO, Worst Case Dynamic Offsets) multiplied by the number of rows to be processed in the conflicts table, which is upper-bounded by the maximum ratio of task deadline over period in the system.

6. Simulation results

We have compared the results of the new analysis for tasks with dynamic offsets and priority schemes with the results obtained using the previous analysis technique for distributed systems based on dynamic offsets without priority schemes [7]. This comparison will be made by means of the results given by Tindell and Clark's technique, which assumes that each task is independent of the others [10]. For this purpose, we have conducted extensive simulations with different task sets whose execution times and periods were generated randomly. Priorities were assigned using the deadline monotonic algorithm with a small random variation that allows us testing cases with different priority schemes. The results of some of these simulations are shown in this section.

The first set of graphs (Figures 8 to 10) compares the response times obtained using Tindell and Clark's technique for independent tasks, R_{indep}, with the response times obtained using algorithm WCDO, R_{WCDO}, and using algorithm WCDOPS, R_{WCDOPS}. In these figures, we show the average ratios $R_{\text{indep}}/R_{\text{WCDO}}$ and $R_{\text{indep}}/R_{\text{WCDOPS}}$ obtained for

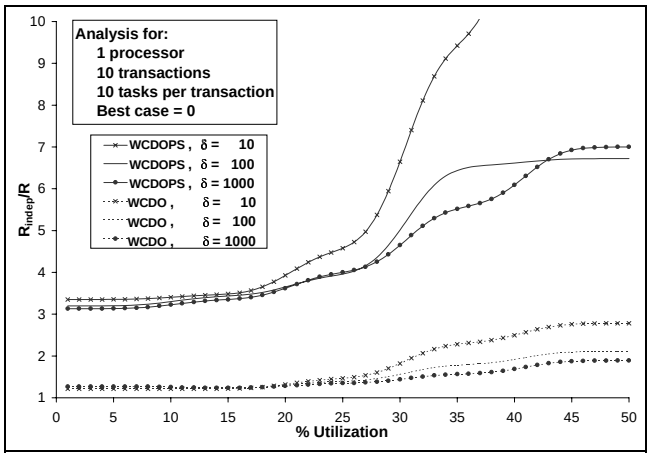

Figure 8. Comparison, one processor, best-case=0

five simulated tasks sets for each point in the graph. The X axis represents processor utilization. Each figure presents the results for three different ratios of the maximum transaction period over the minimum transaction period, $\delta=T_{max}/T_{min}$. Figure 8 shows the results for a set of 10 transactions with 10 tasks per transaction, in one processor, for the case in which the best-case response times are considered negligible, and thus the task offsets are all zero. It can be seen that for normal utilization levels of around 40%, the response times with independent tasks are roughly between 1.7 and 2.5 times larger than in the analysis with dynamic offsets. This result increases to between 6 and 12.5 for the analysis with dynamic offsets and priority schemes. The results with best case response times equal to the task execution times are the same for this case.

Figure 9 shows the results for a similar case, but running on four processors. We can see that as the number of tasks of the same transaction that are in the same processor diminishes, the benefits of the algorithms also diminish. However, these benefits are still significant, with response times between 1.13 and 1.17 times better for 40% utilization in the WCDO algorithm and between 1.65 and 2.23 in the WCDOPS algorithm. Figure 10 shows the results for the same case as Figure 9, except that the best case response time of each task is considered equal to the sum of the execution times of itself and all its predecessor tasks in the same transaction. We can see that the results are better, with response times between 1.45 and 1.77 times better than in the analysis with independent tasks, for a utilization of 40% in the WCDO, that increase until 2 and 3 times for WCDOPS.

The second set of graphs (Figure 11 and Figure 12) compare the maximum schedulable utilization that can be obtained for a given task set using the analysis for independent tasks, algorithms WCDO and WCDOPS with zero best case response times, and algorithm WCDOPS with best case response times equal to the task execution times. The maximum schedulable utilization is obtained by analyzing a system with low utilization, and then increasing its utilization until the system no longer meets its deadlines. The maximum schedulable utilization is taken as the last of the task sets for which the deadlines were met. The simulations have been done for different ratios of deadlines over periods, D_i/T_i.

Figure 11 shows the results for the simulation of a system with 4 processors, 5 transactions and 20 tasks per transaction, with $\delta=T_{max}/T_{min}=100$. Figure 12 shows the results for a similar task system, but with 12 tasks per transaction instead of 20. We can see that from values of $D_i/T_i=3$ and higher, we can get a increase of around 8% more schedulable utilization in the case of 12 tasks, and 15% more in the case of 20 tasks. In Figure 11, for the case $D_i/T_i=4$, that we would consider as normal for a system with 4 processors, we achieve an increase from 15% to 24% of the maximum schedulable utilization, that gives a relative improvement of 60%. In Figure 12, for $D_i/T_i=4$ the maximum is increased from 23% up to 34%, with a relative improvement of 43%. It is also worth mentioning that for

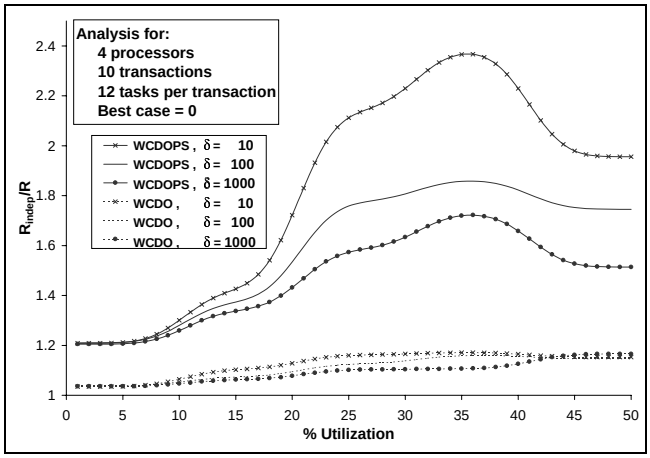

Figure 9. Comparison, four processors, best-case=0

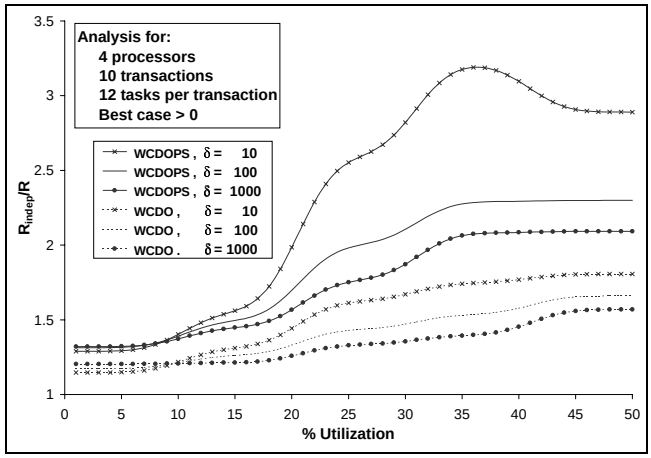

Figure 10. Comparison, four processors, best-case>0

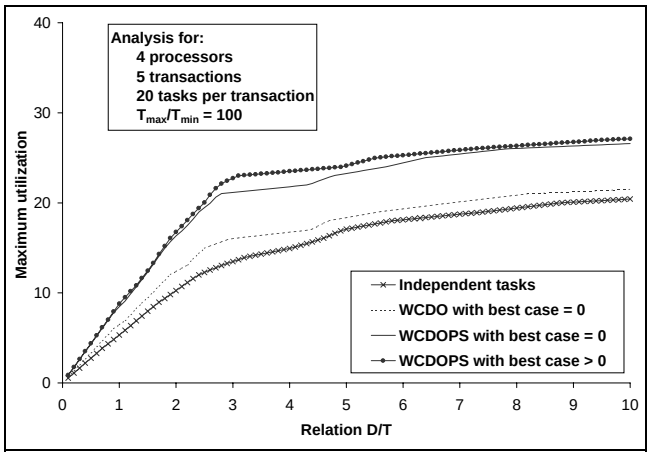

Figure 11. Max. Sched. Utilization, 100 tasks

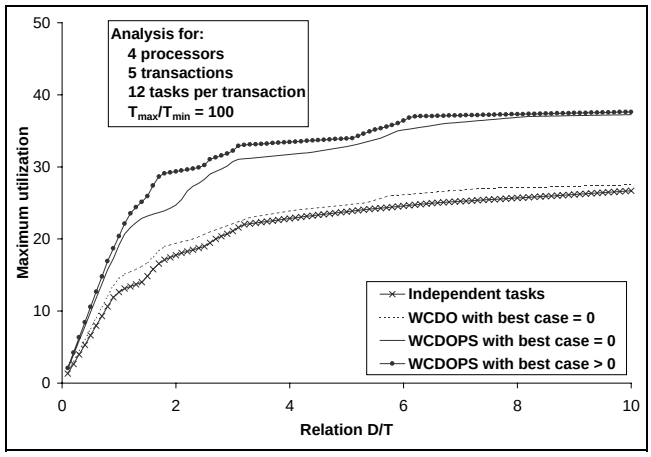

Figure 12. Max.Sched Utilization, 60 tasks

systems with several processors the results are better if we consider best-case response times larger than zero, although it is still possible to get benefits from our new analysis if we consider the best execution times equal to zero.

7. Conclusions

In this paper we have presented improved techniques for the schedulability analysis of tasks with precedence relations in multiprocessor and distributed systems. These techniques are based on the analysis of tasks with dynamic offsets that we had previously developed, which we have improved here by exploiting the precedence relations in a more accurate way, and considering the priority structure of the different tasks. Through simulation results, we have shown that the benefits of the new analysis over the previous analysis techniques for distributed and multiprocessor systems are very high. The response times with the new technique are significantly lower, and the maximum schedulable utilization can be increased; in the examples shown, it was increased by an additional 11% of schedulable utilization.

An implementation of the algorithms WCDO and WCDOPS may be found in *ftp://ftp.deyc.unican.es/pub/real-time*.

References

[1] M. González Harbour, M.H. Klein, and J.P. Lehoczky. "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority". Proceedings of the IEEE Real-Time Systems Symposium, December 1991, pp. 116-128.

[2] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour, "A Practitioner's Handbook for Real-Time Systems Analysis". Kluwer Academic Pub., 1993.

[3] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines". IEEE Real-Time Systems Symposium, 1990.

[4] C.L. Liu, and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". Journal of the ACM, 20 (1), 1973, pp 46-61.

[5] J.C. Palencia, J.J. Gutiérrez García, and M. González Harbour, "On the Schedulability Analysis for Distributed Hard Real-Time Systems". Proceedings of the 9th Euromicro Workshop on Real-Time Systems, Toledo, Spain, June 1997, pp. 136-143.

[6] J.C. Palencia, J.J. Gutiérrez García, and M. González Harbour, "Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems". Proceedings of the 10th Euromicro Workshop on Real-Time Systems, Berlin, Germany, 1998.

[7] J.C. Palencia and M. González Harbour "Schedulability Analysis for Tasks with Static and Dynamic Offsets". Proceedings of the 19 th IEEE Real-Time Systems Symposium, 1998.

[8] J.C. Palencia "Schedulability Analysis of Distributed Real-Time Systems based on Fixed Priorities" PhD thesis (in Spanish), Universidad de Cantabria, Spain, 1999.

[9] L. Sha, R. Rajkumar, and J.P. Lehoczky. "Priority Inheritance Protocols: An approach to Real-Time Synchronization". IEEE Trans. Computers, September 1990.

[10] K. Tindell, and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". Microprocessing & Microprogramming, Vol.50, Nos.2-3, pp.117-134, April 1994.

[11] K. Tindell, "Adding Time-Offsets to Schedulability Analysis", Technical Report YCS 221, Department of Computer Science, University of York, January 1994.