Schedulability Analysis for Tasks with Static and Dynamic Offsets

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

Abstract

In this paper we present an extension to current schedulability analysis techniques for periodic tasks with offsets, scheduled under a preemptive fixed priority scheduler. Previous techniques allowed only static offsets restricted to being smaller than the task periods. With the extension presented in this paper, we eliminate this restriction and we allow both static and dynamic offsets. The most significant application of this extension is in the analysis of multiprocessor and distributed systems. We show that we can achieve a significant increase of the maximum schedulable utilization by using the new technique, as opposed to using previously known worst-case analysis techniques for distributed systems.

1. Introduction

The collection of real-time analysis techniques for fixed-priority systems [1], historically known as rate monotonic analysis [3], represents a mature technology for obtaining guarantees about the timing requirements in hard real-time systems. Although it is possible to get higher utilization levels with dynamic priority scheduling algorithms, fixed priority systems are simpler to analyze and understand, and are supported in standard operating systems and compiler systems.

Rate monotonic analysis (RMA) allows an exact calculation of the worst-case response time of tasks in single-processor real time systems, including the effects of task synchronization [8], the presence of aperiodic tasks, the effects of deadlines before, at or after the periods of the tasks [2], precedence constraints and tasks with varying priorities [4], overhead analysis, etc. However, there are two related areas in which current RMA cannot provide exact solutions to the response times: distributed hard real-time systems, and systems in which tasks suspend themselves. Current techniques for these systems are based on the assumption that all tasks are independent, and thus they lead to pessimistic results [10][6]. If an exact or a less pessimistic technique could be accomplished, this would enable more efficient use of the computing power available in these real-time systems.

Tindell developed in [11] a technique to calculate the worst-case response bound or an upper bound to it, for sets of tasks with static offsets. In his paper, the system is composed of periodic transactions, each containing several tasks. Each task is released after some time, called the offset, elapses since the arrival of the event that triggers the transaction. In [11], the task offsets are restricted to being smaller than the task's periods, and they are static. This is useful in those systems where task activations are timed precisely, at periodic intervals, to avoid the negative effects of jitter. However, a technique to calculate the worst-case response times of tasks sets with offsets could be very valuable to obtain a solution to the problems of task suspension and distributed systems if task offsets could be dynamic, i.e., if they could change from one activation to the next. For example, in distributed systems a task may be released when a previous task completes its execution and a message is received; this release time can vary from one period to the next. In addition, in distributed systems it is common that task deadlines are larger than the periods, and thus it is also very likely that task offsets might become larger than the task periods.

Consequently, in this paper we extend Tindell's analysis of tasks with static offsets in the following ways: we eliminate the restriction of task offsets being smaller than the period; we provide a formal basis that overcomes some defects in [11]; and most important, we extend the technique to cover the case in which task offsets may vary dynamically, and thus we make the technique directly applicable to the analysis of distributed systems and systems

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

in which task suspend themselves. 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 present the analysis for tasks with static offsets. We start with the computational model and discuss existing techniques for solving this problem. Then, we present our extension of Tindell's technique to allow task offsets to be larger than the periods. In Section 3 we extend this technique to the analysis of tasks with dynamic offsets. Then, in Section 4 we show how to apply this analysis to determine the schedulability of a distributed or multiprocessor real-time system. In Section 5 we compare the results obtained with our technique with the results obtained with previously known techniques; we will see that with the new technique the response times are significantly lower. Finally, in Section 6 we give our conclusions.

2. Analysis for Tasks with Static Offsets

2.1. Computational Model

The real-time system that we will consider for the analysis of tasks with static offsets is composed of a set of tasks executing in the same processor, 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. In this section of the paper we will assume that the offset is static, i.e., it does not change from one activation to the next. Each activation of a task releases the execution of one instance of that task, that we will call a *job*.

Figure 1 shows an example of such system: the horizontal axis represents time; down-pointing arrows represent the arrival of the external events associated to each transaction, while up-pointing arrows represent the activation times of each task; and shaded boxes represent task execution. We will assume that each task has its own unique priority, and that the task set is scheduled using a preemptive fixed priority scheduler. Notice that although offsets represent a kind of precedence constraints, in our analysis tasks are activated at a time equal to the arrival of the external event

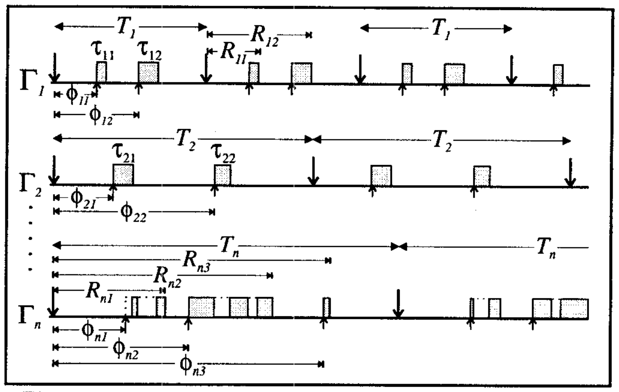

Figure 1. Computational model of a system composed of transactions with static offsets

plus the offset, and they execute at their assigned priority regardless of whether tasks of the same transaction and smaller offsets have finished or not.

Each task will be 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} will be 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, we will allow each task 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 time 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 will allow deadlines to be larger than the period, and thus at each time there may be several activations of the same task pending. We will 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.

We will assume that if tasks synchronize for using shared resources in a mutually exclusive way they will be using a hard real-time synchronization protocol such as the priority ceiling protocol [8]. 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}. Analysis of systems with aperiodic tasks will be addressed in a future paper.

2.2. Current Response Time Analysis Techniques

The RMA analysis technique for tasks with jitter by Tindell and Clark [10], was extended by Tindell [11] to take into account the task offsets, and thus reduce the pessimism of the analysis. He obtained an exact schedulability analysis that was computationally intractable for large task sets, because of the high number of cases that had to be checked, which was exponentially dependent on the number of tasks. However, based upon this technique he developed an approximate upper bound for the worst-case response time, that could be obtained by analyzing a number of cases that was polynomially dependent on the number of tasks. Although the approximate upper bound is pessimistic, it is much closer to the exact solution than the original RMA technique.

However, Tindell's technique was restricted to task offsets being smaller than the associated task periods. In addition, although to our knowledge his results were correct, there were some defects in the development of his analysis; for example, Theorem 1 in [11] did not take into account jitter, and other results were based on this theorem. For these reasons, in this section of our paper we extend his technique to allow task offsets larger than the task periods, and we also formalize the technique via a complete set of proofs. Furthermore, we introduce a different kind of notation, which will help us in a future extension of our technique to further exploit precedence constraints among the tasks of a transaction.

Sun and Liu developed in [9] a technique similar to Tindell's analysis with offsets, and applied it to the analysis in multiprocessor systems, but their technique was restricted to offsets and deadlines smaller than the task periods, and it did not take into account the effects of jitter. Both [11] and this paper handle jitter for task activations, and deadlines larger than the task periods.

2.3. Exact Response-Time Analysis

In this subsection we will extend Tindell's technique for calculating an exact response time analysis of a set of tasks with static offsets [11], to allow offsets larger than the task periods, and we will formalize its development. Although the analysis will be intractable for large task sets because of the large number of cases that need to be considered, it will serve as the basis of the upper-bound approximation that appears in Subsection 2.4.

For building the worst-case scenario for a task τ_{ab} under analysis, 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. For tasks with offsets, we must take into account that the critical instant may not include the simultaneous activation of all higher priority tasks, as it was the case when all tasks were independent. The existence of offsets makes it impossible for some sets of tasks to simultaneously become active.

When analyzing the response time of a particular task, the offset of a higher priority task may be changed by adding or subtracting whole periods of that latter task, without any effects on the response time of the lower priority tasks, since one instance of a task is indistinguishable from another instance. Therefore, in order to simplify the analysis, we will consider a reduced task offset, ϕ_{ij}, which is always within 0 and T_i:

$$\phi_{ij} = \Phi_{ij} \bmod T_i \tag{1}$$

where function *mod* is the usual modulus operation, and where $\lfloor x \rfloor$ is the greatest integer number that is less than or equal to x. This result holds for all higher priority tasks, regardless of whether they belong to the same transaction as the task under analysis, or to different transactions. It also applies when calculating the interference of other jobs of the own task under analysis.

In order to derive the analysis technique, we will try to find out the contribution of each task to the worst-case response time, supposing that we know the time at which the critical instant occurs. Later, we will explore how to calculate the critical instant. Let us focus on the activation pattern of task τ_{ij}, and let us call its phase relation with the critical instant, ϕ, the time interval between the activation of transaction Γ_i that occurred immediately before or at the critical instant, and that critical instant. Notice that $0 \le \phi < T_i$.

In order to calculate the worst-case contribution of τ_{ij} to the response time of lower priority tasks we must categorize each instance of the task into one of the following sets:

- *Set 0*: Activations that occur before the critical instant and that cannot occur inside the busy period even with the maximum jitter delay.
- *Set 1*: Activations that occur before or at the critical instant and that can be delayed by an amount of jitter that causes them to coincide with the critical instant.
- *Set 2*: Activations that occur after the critical instant.

Figure 2 shows two possible scenarios for the alignment

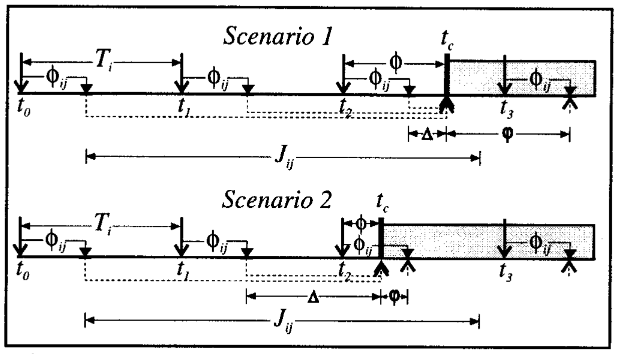

Figure 2. Scenarios for calculating the contribution of task τ_{ij} to the response time of lower priority tasks.

of the transaction T_i's arrival pattern and the critical instant. Scenario 1, in the upper part of the figure, corresponds to the case in which $\phi \geq \phi_{ij}$, and the lower part, Scenario 2, corresponds to $\phi < \phi_{ij}$. Dotted lines represent the actual jitter or delay in the activation time for each instance of the task. Time t_0 corresponds in both scenarios to the first event of Γ_i whose task τ_{ij} may be delayed by jitter until the critical instant, t_c (activations before t_0 would require a delay larger than the maximum jitter to occur at t_c). The event that occurs at t_1 may also be delayed by an amount that makes it coincide with the critical instant. The activation of τ_{ij} associated with the event that arrived at t_2, can be delayed until the critical instant in Scenario 1, but not in Scenario 2, because the offset ϕ_{ij} is larger than the relative phase ϕ between the event arrivals and the critical instant. For scenario 2, the job associated with the event arriving at t_2 must be included in Set 2.

Once the jobs of task τ_{ij} have been categorized into the three sets above, the calculation of the jitter terms that lead to the worst-case contribution of τ_{ij} to the response time of lower priority tasks is done according to the following theorem.

Theorem 1. Given a task τ_{ab} critical instant, t_c, and a phase relation ϕ between the arrival pattern of transaction Γ_i and the critical instant, the worst-case contribution of task τ_{ij} to the response time of τ_{ab} occurs when the activations in Set 1 have an amount of jitter such that they all occur at the critical instant, and when activations in Set 2 have an amount of jitter equal to zero.

Proof: By definition of the busy period, activations in Set 0 are not involved in it; otherwise, since they occur before the critical instant, the busy period would have started earlier.

For activations in Set 1, we must delay them with a jitter amount that causes them to occur inside the busy period. But if this delay causes the activation to occur after the critical instant, it might fall outside the busy period. Thus, to ensure the maximum possible contribution to the busy period, the jitter amount must be such that the activation occurs at the critical instant.

For activations in Set 2, the larger the jitter delay they have, the more probability that the activation occurs outside the busy period. Thus, to ensure the worst possible contribution, the jitter amount for these activations must be zero.□

Under the conditions of Theorem 1, we will now calculate the number of activations of task τ_{ij} that belong to Set 1, and thus that may accumulate at the critical instant. We will call this number n_{ij} (in the example, the upper-part scenario had n_{ij}=3 and the lower-part scenario had n_{ij}=2). To calculate n_{ij}, we will define Δ as the difference in time between the time at which the last activation in Set 1 would have occurred if it had no jitter delay, and the critical instant. In the example of Figure 2, $\Delta = t_c - t_2 + \phi_{ij}$ for Scenario 1, and $\Delta = t_c - t_1 + \phi_{ij}$ for Scenario 2. It can be seen that:

$$\Delta = \begin{cases} \phi - \phi_{ij} & \text{if } \phi \geq \phi_{ij} \\ T_i + \phi - \phi_{ij} & \text{if } \phi < \phi_{ij} \end{cases} \tag{2}$$

or, equivalently:

$$\Delta = (\phi - \phi_{ij}) \bmod T_i \tag{3}$$

The first activation of τ_{ij} in Set 1 corresponds to the event arriving at t_0, which is the first one whose activation may occur at or after the critical instant. Therefore, this is the only activation that simultaneously verifies:

$$t_0 + \phi_{ij} + J_{ij} \geq t_c \tag{4}$$

and:

$$t_0 - T_i + \phi_{ij} + J_{ij} < t_c \tag{5}$$

By looking at Figure 2 we can see that:

$$t_c = t_0 + (n_{ij} - 1)T_i + \phi_{ij} + \Delta \tag{6}$$

and replacing it in the two previous expressions we get:

$$\begin{aligned} t_0 + \phi_{ij} + J_{ij} &\geq t_0 + (n_{ij}-1)T_i + \phi_{ij} + \Delta \\ t_0 - T_i + \phi_{ij} + J_{ij} &< t_0 + (n_{ij}-1)T_i + \phi_{ij} + \Delta \end{aligned} \tag{7}$$

from which we get:

$$n_{ij} - 1 \leq \frac{J_{ij} - \Delta}{T_i} \quad and \quad n_{ij} - 1 > \frac{J_{ij} - \Delta}{T_i} - 1 \tag{8}$$

Given that n_{ij} is an integer number, the solution to the above two expressions is:

$$n_{ij} = \left\lfloor \frac{J_{ij} - \Delta}{T_i} \right\rfloor + 1 \tag{9}$$

In order to determine the effects of activations belonging to Set 2, we need to know the time at which the first of them occurs; the others will occur at periodic intervals after the initial one. We will call φ the time difference between the critical instant and that first activation in Set 2. Given the definition of Δ we have:

$$\varphi = T_i - \Delta \tag{10}$$

We could have used φ in Equation (9) above to obtain:

$$n_{ij} = \left\lfloor \frac{J_{ij}+\varphi}{T_i} \right\rfloor \tag{11}$$

According to Theorem 1, the worst-case contribution of τ_{ij} to a busy period of a lower priority task is equivalent to n_{ij} activations at the critical instant, plus a sequence of periodic activations starting at φ time units after the critical instant. Without loss of generality, let's set the origin of time at the critical instant. Then, the worst-case contribution of task τ_{ij} to the response time of τ_{ab} at time t is determined by:

$$\begin{aligned} W(\tau_{ij},\phi,t) &= n_{ij}(\phi)C_{ij} + \left\lceil \frac{t-\varphi(\phi)}{T_i} \right\rceil C_{ij} = \\ &= \left(\left\lfloor \frac{J_{ij}+\varphi(\phi)}{T_i} \right\rfloor + \left\lceil \frac{t-\varphi(\phi)}{T_i} \right\rceil \right) C_{ij} \end{aligned} \tag{12}$$

with

$$\varphi(\phi) = T_i - (\phi - \phi_{ij}) \bmod T_i \tag{13}$$

The total interference of the tasks of transaction Γ_i on the execution of τ_{ab} is obtained by taking into account the contributions of all higher priority tasks:

$$W(\Gamma_i,\phi,t) = \sum_{\forall j \in hp_i(\tau_{ab})} W(\tau_{ij},\phi,t) \tag{14}$$

where $hp_i(\tau_{ab})$ represents the set of tasks belonging to transaction Γ_i with priority greater than or equal to the priority of τ_{ab}.

Now, we must determine how to calculate ϕ, the phase between the arrival pattern of Γ_i and the critical instant. We will base this calculation on the following theorem:

Theorem 2. The worst-case contribution of transaction Γ_i to a task τ_{ab} critical instant is obtained when the first activation of some task τ_{ik} in $hp_i(\tau_{ab})$ that occurs within the busy period coincides with the critical instant, after having experienced the maximum possible delay, i.e., the maximum jitter, J_{ik}.

Proof. By definition of the busy period, right before the critical instant there are no pending tasks of priority higher than the priority of τ_{ab}. Now suppose that we choose a critical instant that does not coincide with the activation of some task in $hp_i(\tau_{ab})$. Let us focus on the first activation of a task belonging to $hp_i(\tau_{ab})$ that occurs within the busy period, τ_{ik}. If we cause the arrival of the events of Γ_i to occur earlier while keeping the same activation patterns for all its tasks, until task τ_{ik} coincides with the critical instant, all the jobs of tasks belonging to $hp_i(\tau_{ab})$ that were in the busy period continue to be in that same busy period, but we have brought more jobs of those tasks, and perhaps other additional tasks, closer to the busy period, thus increasing the chance of additional interference on task τ_{ab}. Thus by making the first job of τ_{ik} coincide with the critical instant we can only make its contribution worse.

Now, we have to check that the worst-case contribution of transaction Γ_i is obtained when a job of a task τ_{ik} that initiates the critical instant has experienced the worst-case delay, equal to J_{ik}. Let us call I the set of jobs of tasks belonging to $hp_i(\tau_{ab})$ that initiate the busy period, and let us suppose that each of these jobs has a jitter value j_{il} less than the maximum for its associated task, J_{il}. Now let us move back (i.e., earlier in time) the event arrivals of transaction Γ_i, and simultaneously, increase the jitter delay of all the events in I by the same amount of time, so that all these jobs continue to be activated at the same time as before; jitter delays for all other jobs remain unchanged (and thus they are activated earlier). Under these conditions, we will move back the event arrivals until we reach the point when either: a) one of the jobs in I reaches its maximum jitter; or b) when a job in the busy period that did not belong to I gets aligned with the critical instant (because it is activated earlier). In case b), we insert the new job into set I, and we continue the process of moving back the event arrivals of Γ_i, in an iterative manner, until we reach condition a), under which one or more of the activations that start the busy period have experienced their maximum jitter. Notice that during this process, none of the activations that belonged to the busy period has been moved to a point before the critical instant, and thus all the jobs that belonged to the busy period remain in it. However, because the event arrivals of Γ_i now occur earlier, it is possible that jobs which previously occurred after the end of the busy period are now activated inside the busy period, thus making it longer and increasing the response times for the task under analysis, τ_{ab}. Therefore, the theorem follows.□

By applying theorem 2, and supposing that we know that task τ_{ik} is one that originates the critical instant, we can determine the phase between the event arrivals and the critical instant:

$$\phi = (\phi_{ik}+J_{ik}) \bmod T_i \tag{15}$$

Substituting this expression in equation (13) we obtain the phase φ_{ijk} between any task τ_{ij} and the critical instant created with τ_{ik}:

$$\varphi_{ijk} = \varphi(\phi)\,\Big|_{\phi=(\phi_{ik}+J_{ik}) \bmod T_i} = T_i - \big((\phi_{ik}+J_{ik}) \bmod T_i - \phi_{ij} \big) \bmod T_i \tag{16}$$

and applying the properties of the modulus function,

$$\varphi_{ijk} = T_i - \left(\phi_{ik}+J_{ik}-\phi_{ij}\right) \bmod T_i \tag{17}$$

Using this value, we can now obtain the expression of the worst-case contribution of transaction Γ_i when the critical instant is initiated with τ_{ik}. We will call this function $W_{ik}(t,\tau_{ab})$, and we obtain it by replacing (17) in equations (12), (13) and (14):

$$\begin{aligned} W_{ik}(\tau_{ab},t) &= W(\Gamma_i,\phi,t)\,\Big|_{\phi=(\phi_{ik}+J_{ik}) \bmod T_i} = \\ &= \sum_{\forall j\in hp_i(\tau_{ab})} \left(\left\lfloor \frac{J_{ij}+\varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t-\varphi_{ijk}}{T_i} \right\rceil \right) C_{ij} \end{aligned} \tag{18}$$

In order to obtain the worst-case response time of task τ_{ab} we need to apply the above function for all the transactions in the system. The main problem now is that for each transaction Γ_i we need to find the task τ_{ik} with which we create the critical instant. In order to perform an exact analysis, it is necessary to check all possible variations of one task out of every transaction, and choose the variation that leads to the worst response time for the task under analysis.

The number of variations, and thus of different critical instant possibilities that need to be checked, is determined by the number of tasks of priority higher than that of the task under analysis that exist in each transaction in the system. We also have to take into account that the task under analysis itself may originate the critical instant for its transaction. Thus, the total number of variations is:

$$\begin{aligned} N_v(\tau_{ab}) &= \left(N_a(\tau_{ab})+1\right)\cdot N_1(\tau_{ab})\cdot N_2(\tau_{ab}) \;\ldots\; = \\ &= \left(N_a(\tau_{ab})+1\right) \cdot \prod_{\forall i\neq a} N_i(\tau_{ab}) \end{aligned} \tag{19}$$

where $N_i\,(\tau_{ab})$ is the number of tasks belonging to $hp_i(\tau_a)$. Each of the $N_v(\tau_{ab})$ variations is characterized by a tuple v of indexes, one for each transaction. Each index $v(i)$ identifies the task of transaction Γ_i that initiates the critical instant.

For convenience, we will number the jobs of the task under analysis using the letter p, with consecutive numbers ordered according to the activation time that they would have had if they had no jitter. In addition, we will assign the value p=1 to the activation of τ_{ab} that occurs in the interval $(0,T_a]$. This means that the activation that occurred in $(T_a,2T_a]$ gets the value p=2, etc. Similarly, the activation that would have occurred in the interval $(-T_a,0]$ but that was delayed to the critical instant corresponds to p=0, the one in $(-2T_a,-T_a]$ to p=-1, etc. Notice that activations that occur after the critical instant are numbered with positive numbers, while previous activations have values of $p\leq0$.

For each variation v we will obtain the completion time of each of the jobs of τ_{ab} in the busy period. This time, $w^v_{ab}(p)$ is obtained by considering the execution of τ_{ab} together with the interference from all the other tasks in the system:

$$w^v_{ab}(p) = B_{ab}+(p-p^v_{0,ab}+1)C_{ab}+\sum_{\forall i} W_{iv(i)}\left(\tau_{ab},w^v_{ab}(p)\right) \tag{20}$$

where $p^v_{0,ab}$ corresponds to the lowest-numbered job, and is equal to:

$$p^v_{0,ab} = -\left\lfloor \frac{J_{ab}+\varphi_{abv(a)}}{T_a} \right\rfloor + 1 \tag{21}$$

The solution to equation (20) is obtained as in the normal rate monotonic equation [10] by starting from a value of $w^v_{ab}(p)$=0, and iterating until two consecutive iterations produce the same value. This analysis has to be repeated for all the jobs present in the busy period. The length of the busy period, which we will call L^v_{ab}, may be obtained with the following equation:

$$L^v_{ab}=B_{ab}+\left(\left\lceil \frac{L^v_{ab}-\varphi_{abv(a)}}{T_a} \right\rceil - p^v_{0,ab}+1\right)C_{ab}+\sum_{\forall i} W_{iv(i)}\left(\tau_{ab},L^v_{ab}\right) \tag{22}$$

which represents the first instant after the critical instant at which all jobs of τ_{ab} and of all higher priority tasks have been completed. With the length of the busy period, we can obtain the maximum value of p that we need to check:

$$p^v_{L,ab} = \left\lceil \frac{L^v_{ab}-\varphi_{abv(a)}}{T_a} \right\rceil \tag{23}$$

The global response time is obtained by subtracting from the obtained completion time the instant at which the external event that activated the transaction arrived. According to our numbering scheme, the first activation of τ_{ab} after the critical instant corresponds to the value p=1 and, by definition, it corresponds to instant $\varphi_{abv(a)}$. Consequently the p-th activation occurs at $\varphi_{abv(a)} + (p-1)T_a$. Since the task is activated Φ_{ab} time units after the event arrival, the event arrival for each job p occurs at $\varphi_{abv(a)} + (p-1)T_a - \Phi_{ab}$. Therefore, the global worst-case response time for job p is:

$$R^v_{ab}(p) = w^v_{ab}(p) - \varphi_{abv(a)} - (p-1)T_a+\Phi_{ab} \tag{24}$$

Notice that in the above equation we have to use the real offset, Φ_{ab}, instead of the reduced offset ϕ_{ab}, which was used when calculating interference of higher-priority tasks on the task under analysis. To calculate the global worst-case response time for task τ_{ab} we must determine the maximum among all the potential critical instants examined:

$$R_{ab} = \max_{\forall v} \left[\max_{p=p_{0,ab}^{v}..p_{L,ab}^{v}} \left(R_{ab}^{v}(p) \right) \right] \tag{25}$$

By applying the described analysis to each task in the system we can obtain the global worst-case response times and, by comparing them with the deadlines, we can determine whether the system will meet or not its timing requirements. However, although the analysis technique is exact, it represents an NP-complete algorithm in which the number of cases to check grows exponentially with the number of tasks. This means that for most practical problems the algorithm is intractable and cannot be used. For this reason, in the following subsection we will use the upper-bound approximation that appears in [11], in which the number of cases to test is polynomially dependent on the number of tasks, at the price of providing pessimistic results. As it is shown in [11], these results will be much less pessimistic than the ones obtained with the current analysis techniques in [10].

2.4. Upper-Bound Approximation for Worst-Case Analysis

In [11], Tindell developed an approximate method that will enable us to obtain upper bounds for the global worst-case response times in a system composed of transactions with fixed offsets. Although the technique is not exact, the number of cases that need to be checked has a polynomial dependency on the number of tasks, which makes the method applicable even for relatively large systems. If the response times obtained with this method are smaller than the respective deadlines, the method gives guarantees that all timing requirements will be met.

The analysis is based on the exact technique developed in [11] and which we extended in the previous subsection. There, we obtained the equation that calculates the worst-case contribution of a transaction Γ_i on the response time of task τ_{ab} when the critical instant coincides with the activation of task τ_{ik}:

$$W_{ik}(\tau_{ab},t) = \sum_{\forall j \in hp_i(\tau_{ab})} \left(\left\lfloor \frac{J_{ij}+\varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t-\varphi_{ijk}}{T_i} \right\rceil \right) C_{ij} \tag{26}$$

The main problem with that analysis technique is that we don't know which task τ_{ik} must be used to create the worst-case busy period. This caused us to have to check all possible variations. Tindell avoided it by obtaining an upper bound to 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:

$$W_i^*(\tau_{ab},w) = \max_{\forall k \in hp_i(\tau_{ab})} W_{ik}(\tau_{ab},w) \tag{27}$$

Therefore, using this function in the calculation of the response times, we can make sure that the time obtained is an upper bound for the contribution of the tasks of transaction Γ_i and thus it would not be necessary to calculate all the possible variations for k. By using one function like this for each transaction, we can calculate the global worst-case response time for a particular task by checking only a single case.

In order to introduce less pessimism, we will not use that function for the transaction to which the task under analysis belongs, but we will use the original transaction. Consequently, for the analysis we must consider all the possibilities of critical instants created with each of the tasks in the set $hp_a(\tau_{ab})$ plus τ_{ab}; the number of possibilities is small, equal to the number of tasks in $hp_a(\tau_{ab})$ plus one. For a critical instant created with τ_{ac}, the worst-case response time is determined by:

$$\begin{aligned} w_{abc}(p) &= B_{ab}+(p-p_{0,abc}+1)C_{ab} + \\ &+ W_{ac}\left(\tau_{ab},w_{abc}(p)\right) + \sum_{\forall i \neq a} W_i^*\left(\tau_{ab},w_{abc}(p)\right) \end{aligned} \tag{28}$$

As it is shown in [11], this equation can be solved using the traditional RMA iterative method. Parameter $p_{0,abc}$ corresponds to the first activation that occurs at the critical instant:

$$p_{0,abc} = -\left\lfloor \frac{J_{ab}+\varphi_{abc}}{T_a} \right\rfloor + 1 \tag{29}$$

The length of the busy period is calculated as:

$$\begin{aligned} L_{abc} &= B_{ab}+\left(\left\lceil \frac{L_{abc}-\varphi_{abc}}{T_a} \right\rceil - p_{0,abc}+1 \right) C_{ab} + \\ &+ W_{ac}\left(\tau_{ab},L_{abc}\right) + \sum_{\forall i \neq a} W_i^*\left(\tau_{ab},L_{abc}\right) \end{aligned} \tag{30}$$

and from it:

$$p_{L,abc} = \left\lceil \frac{L_{abc}-\varphi_{abc}}{T_a} \right\rceil \tag{31}$$

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

$$R_{abc}(p) = w_{abc}(p) - \varphi_{abc} - (p-1)T_a + \Phi_{ab} \tag{32}$$

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

$$R_{ab} = \max_{\forall c \in hp_a(\tau_{ab}) \cup b} \left[\max_{p = p_{0,abc} .. p_{L,abc}} \left(R_{abc}(p) \right) \right] \tag{33}$$

Notice that this algorithm requires only inspecting a number of possible critical instants equal to the number of tasks in transaction Γ_i that have priorities larger than or equal to the priority of τ_{ab} (and including itself). It is usually a relatively small number, with which we obtain acceptable results, as it can be seen in [11].

3. Analysis for Tasks with Dynamic Offsets

In this section we will extend the analysis to include the case in which the system has tasks with dynamic offsets. As in the case with static offsets, the system is composed of a set of transactions that execute in the same processor. Each transaction Γ_i has a period of T_i and contains a set of m_i tasks with activation offset Φ_{ij}, execution time C_{ij}, and maximum jitter J_{ij}. However, in this case tasks offsets are allowed to vary dynamically, from one activation to the next, within a minimum and a maximum value: $\Phi_{ij} \in [\Phi_{ij,\ min}, \Phi_{ij,\ max}]$.

Dynamic offsets are useful in systems in which tasks suspend themselves or in distributed systems. For example, a task may execute for some time, and then suspend itself to read some data from a disk. Let us suppose that the suspension time is between S_{min} and S_{max}. We would then model this task as a transaction composed of two tasks: task τ_{i1} corresponding to the code before the suspension, and task τ_{i2} to the code after the suspension. The activation time of the second task depends on the completion time of the first task, plus the suspension time, and thus the offset for task τ_{i2} is variable in the interval $\Phi_{i2} \in [R^b_{i1} + S_{min}, R_{i1} + S_{max}]$, where R^b_{i1} and R_{i1} are, respectively, the best-case and worst-case response times of task τ_{i1}. The same kind of effect happens in distributed systems, when a task is activated upon the arrival of a message, which arrives at a variable time within a given time window. Distributed systems are considered in Section 4.

In the analysis for tasks with static offsets that was presented in Section 2, the activation phase represented the minimum interval of time that could exist between the arrival of the external event and the activation of the associated task. On the other hand, the jitter term represented the maximum delay that the task activation could suffer, counted from the arrival of the external event plus the task's offset. Therefore, it is easy to notice that we can model the case in which the offsets may vary as a special case of a system with static offsets, by defining an equivalent static offset Φ'_{ij} and an associated equivalent jitter term J'_{ij}, for each task, in the following way:

$$\begin{aligned} \Phi'_{ij} &= \Phi_{ij,\mathrm{min}} \\ J'_{ij} &= J_{ij} + \Phi_{ij,\mathrm{max}} - \Phi_{ij,\mathrm{min}} \end{aligned} \tag{34}$$

Therefore, the analysis with static offsets can be applied to this equivalent transaction to obtain the worst-case response times. However, in most systems in which task offsets can vary dynamically, their minimum or maximum values, or both, are dependent on the response times of previous tasks in the transaction. For example, in a system with several suspending tasks like the one that was mentioned above, the minimum offset of the task after the suspension was a function of the best-case response time of a previous task in the same transaction, while the maximum offset was a function of the worst-case response of that same task:

$$\begin{aligned} \Phi'_{ij} &= \Phi_{ij,\mathrm{min}} = R^b_{ij-1} + S_{\mathrm{min}} \\ J'_{ij} &= J_{ij} + \Phi_{ij,\mathrm{max}} - \Phi_{ij,\mathrm{min}} = J_{ij} + R_{ij-1} - R^b_{ij-1} + S_{\mathrm{max}} - S_{\mathrm{min}} \end{aligned} \tag{35}$$

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. If task execution times can be arbitrarily small, this lower bound is zero. Otherwise, we can use the task's own best-case execution time. For a more detailed analysis of the best-case response time see [7].

So now our main problem is the calculation of the worst-case response times, which depend on the task offsets, which in turn depend on the worst-case response times. This is a problem similar to the calculation of response times in distributed systems, which depend on the task jitters, while the task jitters themselves depend on the response times. The solution to this problem appears in [10], and consists of starting from an initial value of response times of zero, and iterating over the analysis until a stable solution is achieved. The monotonic dependency of the response on the jitter terms determines the convergence of the method: the larger the jitter terms, the larger the response times, and viceversa.

In our case we will start with an initial value of the response time equal to zero, and thus an initial value for the jitter in the equivalent model of:

$$J'_{ij} = J_{ij} + S_{\mathrm{max}} - S_{\mathrm{min}} \quad \forall i, \forall j \tag{36}$$

Then, 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 (35), and with this new value we recalculate the response times. We continue this calculation in an iterative way until we obtain the same result in two successive iterations, that is:

$$R_{ij}^{(n+1)} = R_{ij}^{(n)} \qquad \forall i \; , \; \forall j \tag{37}$$

We call this algorithm WCDO (worst-case analysis for dynamic offsets). It converges to a solution, if one exists, because of the monotonicity of the worst-case response times given in Eq. (25) with the jitter terms. This dependency is also in accordance with the results of Theorem 2.

4. Analysis of Multiprocessor and Distributed Systems

In multiprocessor and distributed systems it is usual that the system can be modeled with "transactions" composed of several tasks, like in the computational model described in Section 3. For example, in a system following the client-server architecture, a client task is activated by the arrival of an external event, and requests services from one or more servers, perhaps in different processors. This client task can be modeled as a transaction. Each piece of code between the service requests would be modeled as a task. Each portion of execution of a server in another processor is modeled as another task in the same transaction. Each task in the transaction, τ_{ij}, is activated by the completion of the previous task in its transaction τ_{ij-1}.

In a distributed system, the transmission times of the messages through the communications network must also be taken into the analysis. If we use a real-time network that is based on fixed priority messages like the CAN bus or point to point lines [5], we can model the network as if it was another processor, accounting the non-preemptability of the message packets as additional blocking time [6]. Thus, for simplicity, we will only talk about tasks and processors, although one or more of these processors may in fact be modeling a communications network. Other scheduling strategies for the messages in the network can also be adapted to our worst-case analysis.

Consequently, we will model the activities executing in a distributed system as transactions, each composed of a chain of tasks. Each task represents a task or a portion of a task executing in a processor, or a message transmitted through a communications network. The first task in the transaction is activated by the arrival of a periodic external event; let us suppose that this external event has no jitter. We will use a model similar to the one used for tasks that suspended themselves in Section 3, defining an equivalent offset and jitter term of zero for the task that initiates the transaction, and with the following values for each task that does not initiate the transaction:

$$\begin{aligned} \Phi'_{ij} &= \Phi_{ij,\min} = R^{b}_{ij-1} \\ J'_{ij} &= J_{ij} + \Phi_{ij,\max} - \Phi_{ij,\min} = R_{ij-1} - R^{b}_{ij-1} \end{aligned} \tag{38}$$

where R^{b}_{ij-1} is a lower bound to the best-case response time of task, and R_{ij-1} is an upper bound for the worst-case response time.

Evidently, a task executing in a given processor cannot preempt tasks executing in other processors, and thus we must redefine the set $hp_i(\tau_{ab})$ of tasks that can preempt a given task τ_{ab} to contain only tasks that belong to the same processor as τ_{ab}:

$$\begin{aligned} hp_i(\tau_{ab}) = \{\, j \in \Gamma_i \mid & \; priority(\tau_{ij}) \geq priority(\tau_{ab}) \\ & \wedge processor(\tau_{ij}) = processor(\tau_{ab}) \,\} \end{aligned} \tag{39}$$

Using these equivalent offsets and jitter terms we can use the same iterative method that was presented in Section 3, algorithm WCDO, but using equation (38) to calculate the equivalent offsets and jitters, and starting with initial values of $J_{ij}=0$ and $\phi_{ij}=R^{b}_{ij-1}$ for each task. As before, convergence of the iterative algorithm is guaranteed by the monotonic dependency of the response times on the jitter terms.

In order to better understand the technique presented, we will illustrate it with a simple example. Consider the system that appears in Figure 3. Tasks 1, 3 and 5 are simple periodic tasks. Task 2 is a periodic task that suspends itself to request service from task 4. Right before the suspension, task 2 transmits a message, m_1, through the network, which is a serial line. Task 4 is activated at the arrival of this

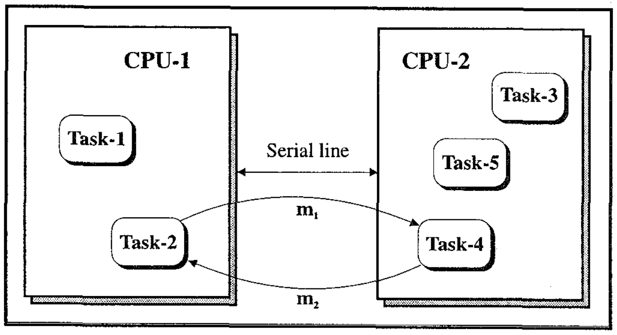

Figure 3. Simple distributed system

	Task	Original	C_i	T_i	D_i	Prio.
Γ_1	τ_{11}	task-1	4	20	20	High
	τ_{21}	task-2_1	20			Low
	τ_{22}	m_1	25			-
Γ_2	τ_{23}	task-4	15	150	150	Med
	τ_{24}	m_2	34			-
	τ_{25}	task-2_2	30			Low
Γ_3	τ_{31}	task-3	5	30	30	High
Γ_5	τ_{51}	task-5	100	200	200	Low

Table I. Timing parameters for the example

message at CPU-2; when it completes its execution it sends message m_2 back through the same serial line. Then, task 2 resumes its execution until completion. Task 2, task 4 and both messages can be modeled as a transaction with five tasks; the timing parameters of all the transactions and tasks are shown in Table I. The scheduling policy used in the network is FIFO; this can be easily modeled by assuming that, when analyzing each message, the other one has higher priority. We will assume that the execution times of each task, and the transmission times of each message are fixed, i.e., there is no difference between the best and the worst-case. This means that we can use the execution or transmission times as the local best-case response times for each associated task, and thus calculate the offset of each task as:

$$\Phi_{ij} = R^b_{ij} = \sum_{k=1..j-1} C_{ik} \tag{40}$$

If we analyze the described system using the conventional analysis techniques in which we assume that each task is independent of the others, we get a response time for transaction Γ_2 of 266 time units, which is well past its end-to-end deadline of 150. However, if we apply the analysis technique described in this section, we obtain a worst-case response time for that same transaction of 145 time units, which makes the transaction schedulable (Table II shows the results of the analysis for the five tasks of transaction Γ_2). The reason for this difference is that in the original analysis the first portion of task 2 is preempted once by the second portion, and viceversa. The same happens for the messages in the network: each one preempts

task	Φ_{2j}	J_{2j}	R_{2j}
τ_{21}	0	0	28
τ_{22}	20	8	53
τ_{23}	45	8	73
τ_{24}	60	13	107
τ_{25}	94	13	145

Table II. Results for transaction Γ_2

the other one once. However, in practice, because of the task offsets, it is not possible that portion 2 of task 2 interferes with portion 1 or viceversa, nor it is possible that the messages interfere with each other. This is correctly taken into account by the analysis with dynamic offsets that we have developed in this paper.

5. Comparison with existing techniques

We have compared the results of the analysis for tasks with dynamic offsets with the results obtained using the current analysis technique for distributed systems, 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 rate monotonic algorithm. The results of some of these simulations are shown in this section.

The first set of graphs (Figures 4 to 6) compares the response times obtained using Tindell and Clark's technique

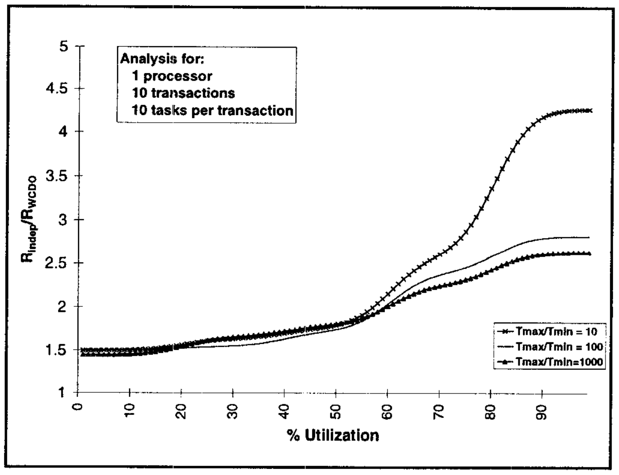

Figure 4. R_{indep}/R_{WCDO}, 1 processor, best=0

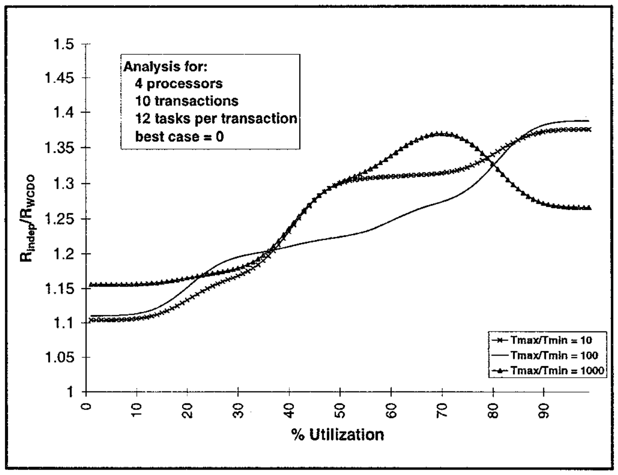

Figure 5. R_{indep}/R_{WCDO}, 4 processors, best=0

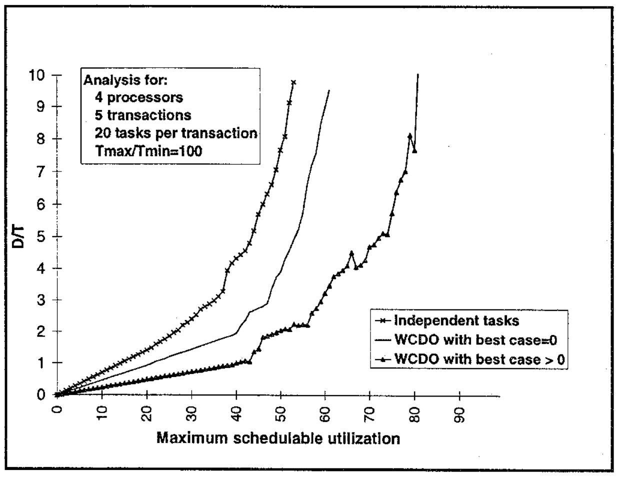

Figure 7. Max. Sched. Utilization, 100 tasks

for independent tasks, R_{indep}, with the response times obtained using algorithm WCDO, R_{WCDO}. In these figures, we show the average ratio R_{indep}/R_{WCDO} obtained for 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, T_{max}/T_{min}. Figure 4 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 70%, the response times with independent tasks are roughly between 2.2 and 2.6 times larger than in the analysis with dynamic offsets. The results with best case response times equal to the task execution times are the same for this case.

Figure 5 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 WCDO algorithm also diminish. However, these benefits are still significant, with response times between 1.27 and 1.37 times better for 70% utilization. Figure 6 shows the results for the same case as Figure 5, 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, in this case, the results are significantly better, with response times between 2 and 2.6 times better than in the analysis with independent tasks, for a utilization of 70%.

The second set of graphs (Figure 7 and Figure 8) compare the maximum schedulable utilization that can be

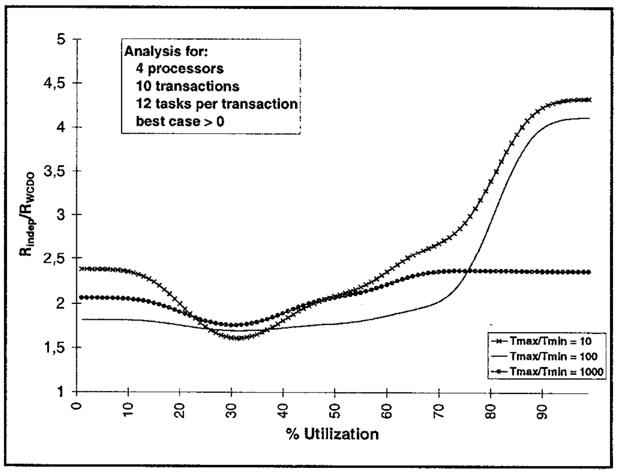

Figure 6. R_{indep}/R_{WCDO}, 4 processors, best>0

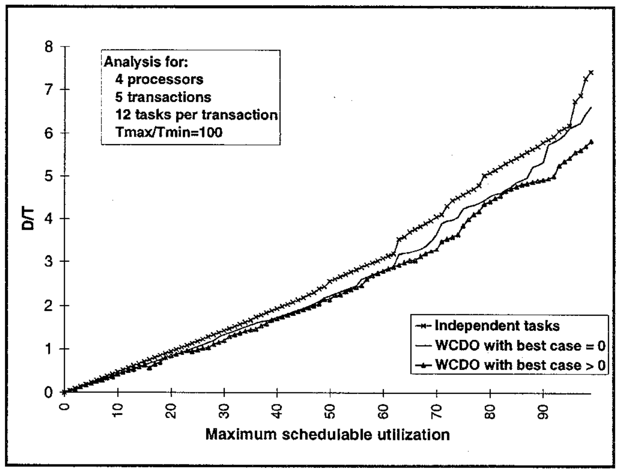

Figure 8. Max. Sched. Utilization, 60 tasks

obtained for a given task set using the analysis for independent tasks, algorithm WCDO with zero best case response times, and algorithm WCDO 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 7 shows the results for the simulation of a system with 4 processors, 5 transactions and 20 tasks per transaction, with T_{max}/T_{min}=100. Figure 8 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=2 and higher, we can get a increase of around 8% more schedulable utilization in the case of 12 tasks, and 25% more in the case of 20 tasks. We can see that, as the number of tasks of the same transaction that execute in the same processor increases, the benefits of the analysis with offsets also increase. It is also worth mentioning that for 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.

6. Conclusions

In this paper we have presented an extension to Tindell's technique for analyzing tasks with static offsets in the context of preemptive fixed-priority scheduling. Our extension consists of allowing the task offsets to be larger than the task periods, and formalizing the development of the technique. In addition, we have extended the technique to the case of dynamic offsets, which vary from one execution to the next. We have shown that the analysis for dynamic offsets is useful for analyzing systems with tasks that suspend themselves, as well as multiprocessor and distributed systems. In all these systems, tasks offsets are dependent on the response time of previous tasks in the same transaction. Through simulation results, we have shown that the benefits of the analysis for dynamic offsets over current analysis techniques for distributed systems are very high. The response times with the new technique are significantly lower. And the maximum schedulable utilization can be increased up to an additional 25% of schedulable utilization.

We are currently working on further extensions of the technique presented in this paper to include aperiodic tasks, as well as further exploiting the effects of the precedence constraints existing in distributed systems and other kinds of systems.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] J.J. Gutiérrez García, and M. González Harbour, "Increasing Schedulability in Distributed Hard Real-Time Systems". Proceedings of the 7th Euromicro Workshop on Real-Time Systems, Odense, Denmark, June 1995, pp. 99-106.

[6] J.C. Palencia Gutiérrez, 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.

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

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

[9] J. Sun and J. Liu, "Bounding the end-to-End Response Time in Multiprocessor Real-Time Systems", Proceedings of the Third Workshop on Parallel and Distributed Real-Time systems, Santa Barbara, CA, 1995.

[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, Dept. of Computer Science, University of York, England, January 1994.