Context-Aware Scheduling Analysis of Distributed Systems with Tree-shaped Task-Dependencies

Rafik Henia, Rolf Ernst
Technical University of Braunschweig
Institute of Computer and Communication Network Engineering (IDA)
D-38106 Braunschweig, Germany
{henia, ernst}@ida.ing.tu-bs.de

Abstract

In this paper we present a new technique which exploits timing-correlation between tasks for scheduling analysis in multiprocessor and distributed systems with tree-shaped task-dependencies. Previously developed techniques also allow capturing and exploiting timing-correlation in distributed systems. However, they are only suitable for linear systems, where tasks cannot trigger more than one succeeding task. The new technique presented in this paper, allows capturing timing-correlation between tasks in parallel paths in a more accurate way, enabling its exploitation to calculate tighter bounds for the worst-case response time analysis for tasks scheduled under a static priority preemptive scheduler.

1. Introduction

For simplicity, most formal scheduling analysis techniques ignore correlations between task execution times or communication timing. This avoids the growing analysis complexity, in particular when it comes to heterogeneous multiprocessor systems. However, such correlations can have a large influence on system timing as has been shown for special system topologies [6] [4]. This paper extends the analysis to more general structures.

Observe the system in figure 1. The system consists of five tasks mapped on three resources. Due to the data-dependency between the tasks, their activating events are time-correlated. We call the information about such correlation *inter event stream context* [1]. However, a typical scheduling analysis would ignore the available inter event stream context and would assume that all tasks are independent and that in the worst-case they are activated simultaneously [7]. This may lead to a greater calculated maximum number of interrupts of a lower-priority task by higher-priority tasks, resulting in a longer calculated worst-case response time of the lower priority task.

Methods exploiting inter event stream contexts for the worst-case response time calculation already exist. However, they are either limited to linear systems and tasks in single paths [6] [4], e. g. the correlation between T_2 and T_5 on R_2, or enable to capture the complete timing-correlation between tasks in parallel paths [2] [3], e. g. the correlation between T_3 and T_4 on R_3, in an accurate way. All this methods only consider the external events (produces by $source$) as references to capture timing-correlation between tasks.

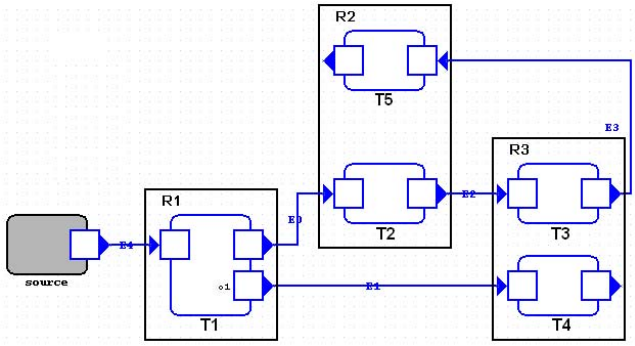

Figure 1. Distributed system with timing-correlation between tasks in parallel paths

In this paper, we present a new technique to capture the inter event stream context information for tasks in parallel paths and exploit it for the worst-case static priority scheduling analysis to calculate tighter response time bounds.

In the following section, we will more deeply review the existing approaches from literature to exploit inter event stream contexts for worst-case response time calculation in distributed systems. In Section 3, we introduce our computational model, then we present some inter event stream context preliminaries in section 4. In section 5, we show the limits of existing techniques in exploiting inter event stream contexts for tasks in parallel paths and introduce the idea of relative offset and relative jitter. An algorithm for the worst-case response time calculation considering relative offset and jitter information is presented in section 6. Experiments are carried out in section 7. We interpret the experimental results, before we draw our conclusions.

2. Related Work

Inter event stream contexts capture the timing correlation between events in a way that can be exploited by schedul-

1530-1591/05 $20.00 © 2005 IEEE

ing analysis. Tindell introduced this idea for tasks scheduled by a static priority preemptive scheduler [6]. In his paper, each set of time-correlated tasks is grouped into a so called *transaction*. Each transaction is activated by a periodic sequence of external events. Each task belonging to a transaction is activated when a relative time, called *offset*, elapses after the arrival of the external event. An activation of a task releases the execution of one instantiation of that task, called *job*. However, Tindell's technique did not allow offsets to be larger than the transaction period.

Tindell's work was later generalized by Palencia and Harbour [4]. They presented the WCDO (Worst Case Dynamic Offsets) algorithm which extends the analysis presented by Tindell, by allowing the task offsets to be larger than the transaction period and extending the technique for distributed systems to dynamic offsets, which vary from one job to another. In [5], Palencia and Harbour presented a new analysis technique for tasks with precedence relations in distributed systems. The presented technique called WCDOPS (Worst Case Dynamic Offsets with Priority Schemes) extended the WCDO algorithm by exploiting precedence relations among tasks during analysis. However, the WCDOPS algorithms only took into account tasks in linear transactions, where each task is allowed to have at most one successor.

In recent works, Redell extended the WCDOPS algorithm by considering precedence relations between tasks in so called tree-shaped transactions [2] [3], by allowing tasks to have more than one successor. However even though the algorithm he proposed (the WCDOPS+ algorithm) allows exploiting the inter context information for tasks in parallel paths, it was based on the inter context capturing technique presented by Palencia and Harbour, which was developed for linear systems. Therefore, as we will show, not all available timing-correlation was exploited.

3. Computational Model

The model that we consider is composed of tasks executing in a distributed system consisting of computation and communication resources. Tasks are allowed to have more than one immediate successor. Each task is assumed to have exactly one input and is activated due to one activating event. After finishing its execution, a task produces exactly one event at each of its outputs. The possible timing of events is described using *event models*. Event models are described using two parameters: the *period* and the *jitter*, noted P and J. These parameters state that each event generally occurs periodically with a period P, but that it can jitter around its exact position within a jitter interval J. If the jitter is larger than the period, then two or more events can occur at the same time, leading to *bursts*. Tasks are also assigned priorities. An execution of a lower priority task can be interrupted by the execution of a higher priority task mapped on the same resource. The response time R of a task is defined as the difference between its completion and its activation time.

4. Transactions

In this section, we review preliminaries about capturing the inter event stream context information and exploiting it for the worst-case response time calculation, as presented in [6] and [4]. Each set of time-correlated tasks in the distributed system is grouped into one transaction. In addition, each task belonging to a transaction is identified by an offset parameter which indicates the earliest activation time after the arrival of the associated external event activating the transaction. In the following, we call this offset *global offset*.

To calculate the worst-case response time of a lower priority task T_l, we must calculate the maximum contribution from all the transactions to its *busy period*. The busy period of T_l is a time-interval during which the resource is busy processing T_l or another task from $hp(T_l)$, where $hp(T_l)$ is the set of higher or equal priority tasks sharing a same resource with T_l. The instant that starts the busy period is called *critical instant* and is noted t_c. In [6] and [4], it was shown that the maximum contribution of a transaction Γ to the busy period is obtained when the critical instant t_c coincides with the activation time of some task $T_c \in hp_\Gamma(T_l)$ ($hp_\Gamma(T_l)$ is the set of tasks belonging to $hp(T_l)$ and Γ) when T_c is delayed by its maximum jitter. In order to perform an exact analysis, it is necessary to check all possible critical instants created by all tasks from $hp_\Gamma(T_l)$ and choose the one that leads to the worst-case response time of T_l.

Figure 2 shows execution timing of a task T_i which belongs to $hp_\Gamma(T_l)$. The downward arrows indicate the external events activating the transaction. The upward arrows indicate the offset Φ_i of T_i. Assuming that T_i is activated by the event model (P_i, J_i), its activation can occur between the instants $t_0 + \Phi_i$ and $t_0 + \Phi_i + J_i$, where t_0 is the instant at which the associated external event arrived.

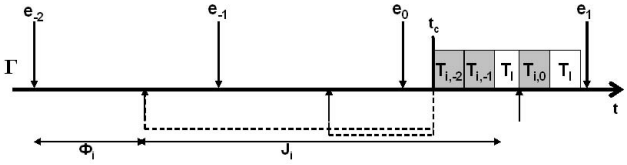

Figure 2. Transaction with executions of T_l and jobs of T_i

The maximum contribution of T_i to the worst-case busy-period of T_l is obtained, as shown in figure 2, when T_i is activated if possible at, or as soon as possible after t_c.

5. Relative Offset And Relative Jitter

5.1 Problem Formulation

Observe the system modeled in figure 1. We assume static priority scheduling on R_2 and R_3. The priorities are assigned as follows: $T_2 > T_5$ and $T_3 > T_4$. The core execution times, i.e. assuming no interrupts, are assumed to be [2,8] for T_1 and [2,2] for all other tasks. T_1 is assumed to be activated periodically by events sent by the source task $source$. Let the

activating event model of T_1 be $(P_1 = 10, J_1 = 0)$. At the end of each execution, T_1 produces exactly one event at each of its outputs. Due to the data-dependency between T_1, T_2, T_3, T_4 and T_5, we group all tasks in one transaction. Without loss of generality, we assume that T_1 has a global offset $\Phi_1 = 0$. In the following, we will focus on the worst-case response time of task T_4.

Table 1 shows the input event models and offsets of T_3 and T_4 after having analyzed the resources R_1 and R_2. Note that the jitter is due to the response time variation of T_1.

task	input event model	offset
T_3	$(P_3 = 10, J_3 = 6)$	$\Phi_3 = 4$
T_4	$(P_4 = 10, J_4 = 6)$	$\Phi_4 = 2$

Table 1. Offsets and input event models of T_3 and T_4

The worst-case response time of the lower priority task T_4 calculated by the WCDO [4] and WCDOPS+ [2] [3] algorithms is $R_4^w = 4$. This worst case response time, as shown in the gantt-chart in figure 3, is obtained when T_4 starts the critical instant, i.e. that T_4 is activated after having arrived as late as possible. The maximum contribution of T_3 is obtained by delaying its activation by 4 time units which causes it to coincide with the critical instant. Therefore, one interrupt of T_4 by T_3 is calculated.

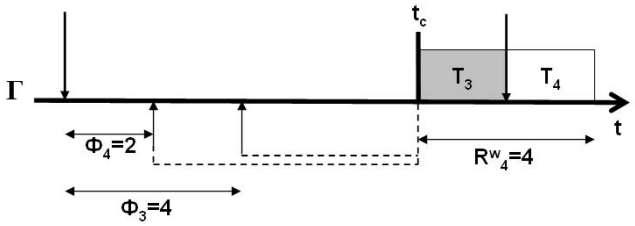

Figure 3. worst-case response time calculation of T_4 using WCDO and WCDOPS+

Now, let us take a closer look on our system. Since T_1 produces one event at each output after each execution, T_2 and T_4 are always activated simultaneously. As both tasks share different resources, there is no conflict between their execution requests. Therefore, since T_3 is activated after T_2 finishes its execution (which takes 2 time units), an activation of T_3 always occurs 2 time units after an activation of T_4. As no former activation of T_3 can be delayed by an amount of jitter that cause it to delay or interrupt the execution of T_4, the true worst-case response time of T_4 is $R_4^w = 2$.

The pessimistic worst-case response time calculation of the WCDO and WCDOPS+ is due to the fact that these algorithms only consider the external events (here events produced by $source$) as references to capture timing correlations between tasks in the system. I.e. they ignore that, other references in the system could lead to an exacter calculation. In the example above, since events at both outputs of T_1 are produced simultaneously, it would be more accurate to consider the completion time of T_1 as a reference to capture the timing correlation between the tasks in the parallel paths starting at T_1.

5.2 Relative Offset and Relative Jitter Concept

To capture the timing correlation between tasks in parallel paths, we introduce the concept of *relative offset and relative jitter*.

Definition 1 (relative offset). *A task T_i is said to be activated after an offset $\Phi_i(T_r)$ relative to a reference task T_r, if T_i is activated at the earliest when a relative time $\Phi_i(T_r)$ elapses after the completion time of T_r. $\Phi_i(T_r)$ is called offset of T_i relative to T_r.*

Definition 2 (relative jitter). *The activation of a task T_i relative to the completion time of a reference task T_r can vary within a jitter interval of length $J_i(T_r)$. $J_i(T_r)$ is called jitter of T_i relative to T_r.*

The offset and jitter information of a task T_i relative to task T_r shall be denoted: (T_r, $\Phi_i(r)$, $J_i(r)$).

The relative offset and relative jitter calculation extends the global offset and jitter calculation by allowing any reference task not just external sources. I.e. the relative offset corresponds to the minimum path latency starting from the reference task and the relative jitter corresponds to the difference between the maximum and the minimum path latency starting from the reference task.

In the example in figure 1, since T_2 and T_4 are activated immediately after the execution completion of T_1, the offset and jitter information of T_2 and T_4 relative to T_1 is: (T_1, 0, 0). Since T_3 is activated immediately after T_2 finishes its execution, which takes 2 time units in both best- and worst-case, the offset and jitter information of T_3 relative to T_1 is: (T_1, 2, 0).

Depending on the system topology a task may have an offset and jitter relative to several reference tasks. This is shown in figure 4 which represents an extension of the system in figure 1. T_5, T_6 and T_8 have an offset and jitter relative to T_1 and T_3 while T_7, T_9 and T_{10} have an offset and jitter relative to T_1 and T_4.

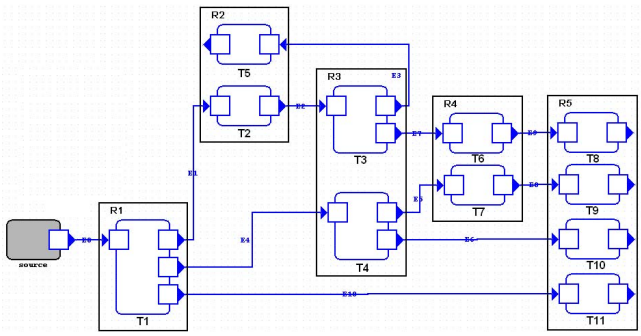

Figure 4. System showing tasks having offset and jitter relative to several references

6. Worst Case Response Time Analysis

In this section, we derive the worst-case response time calculation for a lower priority task T_l considering the relative

offset and jitter information for higher or equal priority tasks belonging to the same transaction Γ.

The worst-case scenario for the task under analysis is obtained by constructing a critical instant t_c that leads to the worst-case busy period. Let us assume that t_c is started by some task $T_c \in hp_\Gamma(T_l)$. Without loss of generality, we set the origin of time at t_c. We will first show the maximum contribution of tasks belonging to $hp_\Gamma(T_l)$ to the busy-period, by considering global offsets only. Then we will additionally consider the relative offset and jitter.

We assign indexes to each external event, e, activating the transaction Γ as shown in figure 2. The first external event that occurs before or at t_c is denoted e_0. Previous external events are assigned negative indexes. Following external events are assigned positive indexes. For each task T_i belonging to $hp_\Gamma(T_l)$, each job is assigned the index of the associated external event. A job of T_i with an index k is denoted $T_{i,k}$. The activation instant of $T_{i,k}$ is denoted $t_{i,k}$.

The earliest job of T_i that can be delayed enough to be activated at the critical instant is denoted n_i. In Figure 2, n_i is -2. In [6] and [4], it was proven that the maximum contribution of a job $T_{i,k}$, with $k \geq n_i$, to the busy period is obtained by delaying its activation by a certain amount of jitter to coincide with t_c, or to activate it without any delay if its earliest activation time occurs after t_c: e.g. in figure 2, the maximum contribution of T_i to the busy period is obtained by activating $T_{i,-1}$ and $T_{i,-2}$ at t_c and all the following jobs without any delay. Note that jobs with an index smaller than n_i are not considered since they cannot be delayed enough to be activated at t_c.

So far, to calculate the maximum contribution of a job $T_{i,k}$ to the worst-case busy period, we determined its activation instant depending on its global offset and jitter only. However, if a relative offset and jitter information is available, an other dependency could exist between $T_{i,k}$ and jobs triggered by the associated external event e_k, i.e. jobs having the same index k, and belonging to other tasks from $hp_\Gamma(T_l)$. Recall the example in section 5.1. By exploiting the available offset and jitter information relative to T_1, we found out that an activation of T_4 precedes the activation of T_3 triggered by the same associated external event, by 2 time units. I.e. although each job can be delayed to coincide with t_c, it is impossible to activate them simultaneously at t_c.

Let T_j be a task which also belongs to $hp_\Gamma(T_l)$. We assume that both T_i and T_j have an offset and jitter information relative to a reference task T_r. Let (T_r, $\Phi_i(T_r)$, $J_i(T_r)$) be this information for T_i and (T_r, $\Phi_j(T_r)$, $J_j(T_r)$) for T_j. Consider the jobs $T_{i,k}$ and $T_{j,k}$, with $k \geq n_i$, $k \geq n_j$ and $k \leq 0$. Without loss of generality, let us assume that due to the available relative offset and jitter information, the activation of $T_{i,k}$ always precedes the activation of $T_{j,k}$. If we assume a maximum contribution of $T_{i,k}$ to the busy period by activating it at t_c, $T_{j,k}$ will be activated after t_c, and thus its activation may occur outside the busy period. On the other hand, if we assume a maximum contribution of $T_{j,k}$ to the busy period by activating it at t_c, $T_{i,k}$ must be activated before t_c and thus, is not considered for the busy period. Figure 5 shows both activation scenarios for $T_{i,k}$ and $T_{j,k}$. In general, when considering the available relative offset and jitter information, and in order to obtain the maximum contribution to the busy period of jobs belonging to different tasks and having the same index, we have to consider all activation scenarios. In each scenario, a maximum contribution to the busy period is assumed for one job by activating it at t_c. Depending on the activation instant of this job and all available relative offset and jitter information, activation instants for the other jobs are calculated. Note that this is very similar to the global offset exploitation concept, where it is necessary to check all possible critical instant construction scenarios and choose the one that leads to the worst-case response time [4]. In the following, the activation scenario of jobs triggered by the external event e_k is denoted A_k. When a maximum contribution to the busy period is assumed for a job with index k, we say that this job determines the activation scenario A_k.

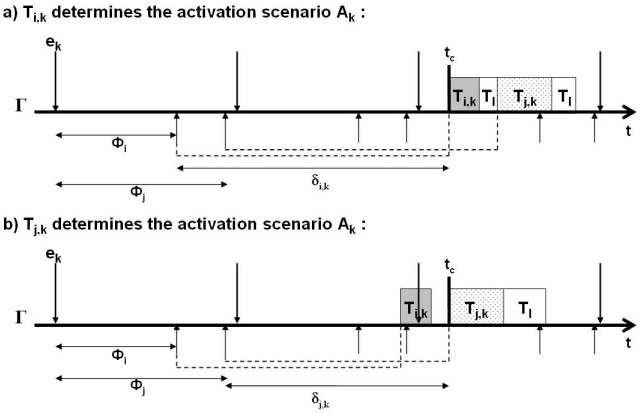

Figure 5. Activation scenarios for $T_{i,k}$ and $T_{j,k}$

In section 5, we stated that each task may have an offset and jitter relative to several reference tasks. When constructing an activation scenario, we choose a reference task which is common to all tasks on the same resource and having relative offset and jitter information. If there are several common reference tasks, we choose the most "recent" task as reference. In the example in figure 4, assuming that we are constructing an activation scenario for T_9 and T_{10} only, we choose the task T_4 as reference task among the set of common reference tasks T_1, T_4. When constructing an activation scenario for T_9, T_{10} and T_{11}, the only common reference task is T_1.

After having constructed all activation scenarios for jobs having the same index, we still need to combine them with activation scenarios for jobs having other indexes. Assume that $k-1 \geq n_i$ and $k-1 \geq n_j$, i.e. the activations of $T_{i,k-1}$ and $T_{j,k-1}$ can also be delayed to coincide with t_c. Figure 6 shows the activation scenarios A_{k-1} and A_k involving jobs of T_i and T_j, assuming a higher priority for T_i. Note that since an activation of a job cannot precede the activation of

an anterior job belonging to the same task, $T_{i,k-1}$ and $T_{j,k}$ cannot respectively determine the activation scenarios A_{k-1} and A_k at the same time. In general, if a job of some task T_s determines an activation scenario, all activation scenarios having higher indexes can only be determined by jobs either belonging to T_s or belonging to tasks that precede T_s. This allows a reduction of the number of activation scenarios combinations we need to check. Note that the process of combining activation scenarios is also very similar to the process of combining critical instant candidates belonging to different transactions, when exploiting global offsets [4].

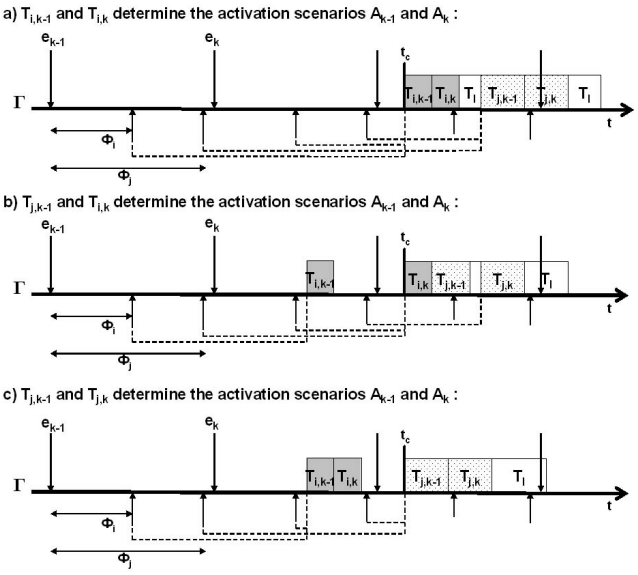

Figure 6. Activation scenarios A_{k-1} and A_k involving jobs of T_i and T_j

In the following, we drop the assumption that an activation of T_i always precedes the activation of T_j. Let us now assume that $T_{i,k}$ determines the activation scenario A_k by activating $T_{i,k}$ at the instant $t_{i,k} = t_c$. Let $\delta_{i,k}$ be the delay needed by the activation of $T_{i,k}$ to occur at t_c (see $\delta_{i,k}$ in figure 5). The earliest occurrence of $t_{j,k}$, $t_{j,k}^{min}$, can be expressed by the following equation:

$$t_{j,k}^{min} = t_c + \Phi_j(T_r) - \Phi_i(T_r) - min(\delta_{i,k}, J_i(T_r)) \quad (1)$$

Proof. Let us first calculate the earliest completion time of $T_{r,k}$ as a function of $t_{i,k}$. The activation of $T_{i,k}$ occurs after an offset $\Phi_i(T_r)$ after the completion time of $T_{r,k}$ and can experience a maximum delay of $J_i(T_r)$. On the other hand, this delay cannot exceed $\delta_{i,k}$. Therefore, the maximum delay experienced by $T_{i,k}$, relative to $T_{r,k}$, corresponds to $min(\delta_{i,k}, J_i(T_r))$. The earliest completion time of $T_{r,k}$ occurs consequently at the instant $t_{i,k} - \Phi_i(T_r) - min(\delta_{i,k}, J_i(T_r))$. Now we can calculate the earliest occurrence of $t_{j,k}$ as a function of $t_{i,k}$. $T_{j,k}$ is activated at the earliest after an offset $\Phi_j(T_r)$ after the completion time of $T_{r,k}$. Therefore, the earliest activation instant of $T_{j,k}$ is $t_{i,k} + \Phi_j(T_r) - \Phi_i(T_r) - min(\delta_{i,k}, J_i(T_r))$. Since $t_{i,k} = t_c$, equation 1 holds. □

The latest occurrence of $t_{j,k}$, $t_{j,k}^{max}$, can be expressed by the following equation:

$$\begin{aligned} t_{j,k}^{max} &= t_c + \Phi_j(T_r) - \Phi_i(T_r) + J_j(T_r) \\ &\quad - max(0, \delta_{i,k} + J_i(T_r) - J_i) \end{aligned} \quad (2)$$

Proof. Let us first calculate the latest completion time of $T_{r,k}$ as a function of $t_{i,k}$. The activation of $T_{i,k}$ occurs at the earliest after an offset $\Phi_i(r)$ after the completion time of $T_{r,k}$, i.e. without experiencing any delay. The delay $\delta_{i,k}$ is thus, assumed to be generated only due execution time variations of $T_{r,k}$ and jobs of tasks preceding T_r and having the index k. However, since this delay can not exceed $J_i - J_i(T_r)$, the minimum delay experienced by $T_{i,k}$ relative to $T_{r,k}$ corresponds to $max(0, \delta_{i,k} + J_i(T_r) - J_i)$. Therefore, the latest completion time of $T_{r,k}$ occurs at the instant $t_{i,k} - \Phi_i(T_r) - max(0, \delta_{i,k} + J_i(T_r) - J_i)$. Now we can calculate the latest occurrence of $t_{j,k}$ as a function of $t_{i,k}$. $T_{j,k}$ is activated at the latest after an offset $\Phi_j(T_r)$ and a maximum delay $J_i(T_r)$ after the completion time of $T_{r,k}$. Therefore, the latest activation instant of $t_{j,k}$ is $t_{i,k} + \Phi_j(T_r) - \Phi_i(T_r) + J_j(T_r) - max(0, \delta_{i,k} + J_i(T_r) - J_i)$. Since $t_{i,k} = t_c$, equation 2 holds. □

Now we can calculate the instant $t_{j,k}$ which leads to a maximum contribution of $T_{j,k}$ to the busy period, under the assumption that $T_{i,k}$ determines A_k. Since the activation of $T_{j,k}$ cannot precede the activation of $T_{j,k-1}$, the maximum contribution of $T_{j,k}$ to the busy period is obtained when $t_{j,k} = max(t_c, t_{j,k-1})$. In addition, $t_{j,k}$ belongs to the interval $[t_{j,k}^{min}, t_{j,k}^{max}]$. Therefore, we have to distinguish following cases:

- $max(t_c, t_{j,k-1}) < t_{j,k}^{min}$: in this case, the maximum contribution of $T_{j,k}$ to the busy period is obtained when it is activated as soon as possible after the instant $max(t_c, t_{j,k-1})$. I.e. $t_{j,k} = t_{j,k}^{min}$. In the example in section 5.1, the maximum contribution of $T_{3,0}$ to the busy period is obtained when $t_{3,0} = t_c = 0$. However, since $t_{3,0}^{min} = 2$ under the condition that $T_{4,0}$ determines the activation scenario A_0, $t_{3,0} = t_{3,0}^{min} = 2$.
- $max(t_c, t_{j,k-1}) > t_{j,k}^{max}$: We know that $t_{j,k}^{max}$ is greater or equal $t_{j,k-1}$. Consequently, $max(t_c, t_{j,k-1}) = t_c$. Therefore, the instant $t_{j,k}$ occurs before t_c. I.e. $T_{j,k}$ does not contribute to the busy period. In the example in section 5.1, the maximum contribution of $T_{4,0}$ to the busy period is obtained when $t_{4,0} = t_c = 0$. However, since $t_{4,0}^{max} = -2$ under the condition that $T_{3,0}$ determines the activation scenario A_0, $t_{4,0}$ occurs before t_c.
- $max(t_c, t_{j,k-1}) \in [t_{j,k}^{min}, t_{j,k}^{max}]$: in this case, the maximum contribution of $T_{j,k}$ to the busy period is obtained when $t_{j,k} = max(t_c, t_{j,k-1})$.

Now after having considered jobs with indexes lower or equal 0, we will calculate the activation instants of jobs with

indexes greater than 0. In the following, we show that the activation of $T_{j,1}$ cannot be delayed by the activation of previous jobs and thus, the maximum contribution to the busy period of jobs with indexes greater than 0 is obtained when they are activated, as before, without experiencing any delay.

The activation of $T_{j,1}$ cannot occur before the instant $t_c + \Phi_j$. I.e. $t_{j,1}^{min} \geq t_c + \Phi_j$. On the other hand, as stated above, depending on the relative offset and jitter information, $t_{j,k}$, with $n_j \leq k \leq 0$, occurs either before t_c, at t_c, at $t_{j,k-1}$ or at $t_{j,k}^{min}$. We show that $t_{j,k} \leq t_c + \Phi_j$:

- $t_{j,k} \leq t_c$: it is obvious that $t_{j,k} \leq t_c + \Phi_j$
- $t_{j,k} = t_{j,k}^{min}$: using equation 1, we can state that $t_{j,k}^{min} \leq t_c + \Phi_j(T_r)$. Therefore, since $\Phi_j \geq \Phi_j(T_r)$, $t_{j,k}^{min} \leq t_c + \Phi_j$.
- $t_{j,k} = t_{j,k-1}$: since $t_{j,k-1}$ also occurs either before t_c, at t_c, at $t_{j,k-2}$ or at $t_{j,k-1}^{min}$, we can replace $k-1$ by k and repeat the same process described above.

Since no job having an index lower or equal 0 can delay the activation of $T_{j,1}$, jobs with indexes greater than 0 can be considered activated as before, without any delay. Therefore, there is no need to exploit the potential relative offset and jitter information for these jobs.

As explained above, to exploit relative offset and jitter for the worst-case response time calculation we need to consider all combinations of activation scenarios with indexes lower or equal 0. Since for each task, the number of jobs that can be delayed to coincide with t_c is bounded, the relative offset and jitter exploitation only adds a polynomial number of cases to check to the global offset exploitation algorithm WCDO.

7. Experiments

We have performed a large number of experiments using randomly generated systems with tree-shaped task-dependencies. Tasks mapping, event models, core execution times and priorities were also assigned randomly. We have compared the results obtained using our technique considering relative offset an jitter with the results obtained by the inter event stream context blind analysis (i.e. without considering timing-correlation between tasks), WCDO, and WCDOPS+.

Figure 7 shows the response time average ratios $R_{blind}/R_{relative}$, $R_{WCDO}/R_{relative}$ and $R_{WCDOPS+}/R_{relative}$ as function of the system utilization. The results show that a large improvement can be obtained due to the relative offset and jitter exploitation: up to 66% compared to the inter event stream context blind analysis, up to 57% compared to the WCDO and up to 41% compared to WCDOPS+. It is also interesting to note that in general, a larger improvement is obtained for large system utilization. This is due to the fact that a large system utilization leads to higher calculated worst-case response times. This in turn leads to larger task global jitters on which, the response times themselves depend. When considering relative jitters, the effect of global jitters on response times calculation can be reduced and thus, lower response-times are calculated.

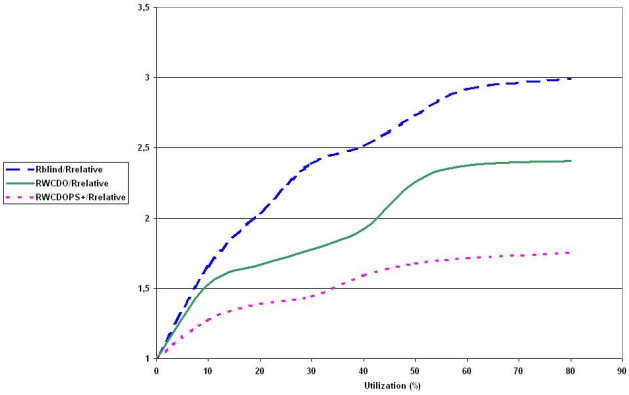

Figure 7. Response time average ratios as a function of utilization

8. Conclusion

In this paper we presented a new technique to capture timing-correlation between tasks in distributed systems with tree-shaped task-dependencies. We have seen that considering the external system events activating tasks as unique timing reference does not allow to capture the complete existing timing-correlation between tasks in parallel paths. Our solution consists in considering tasks with several successors as additional timing-references. We also developed an algorithm to exploit this approach for the worst-case response time analysis under a static priority scheduler. Through our experiments, we showed that our technique allows to calculate considerably tighter bounds compared to other techniques. We consider that our approach is an important extension of the collection of analysis techniques exploiting timing-correlation between tasks in distributed systems.

References

[1] M. Jersak, R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded system design. In *Proc. of Design, Automation and Test in Europe (DATE'04)*, Paris, France, Mar. 2004.

[2] O.Redell. Accounting for precedence constraints in the analysis fo fixed priority scheduled tasks. Technical Report 2003:4, TRITA-MMK, 2003.

[3] O.Redell. Analysis of tree-shaped transactions in distributed real time systems. In *Proc. of 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, June 2004.

[4] J. C. Palencia and M. G. Harbour. Schedulablilty analysis for tasks with static and dynamic offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS98)*, 1998.

[5] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulablilty analysis of distributed real-time systems. In *Proc. 20th Real-Time Systems Symposium*, 1999.

[6] K. W. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Univ. of York, 1994.

[7] K. W. Tindell. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, Mar 1994.