Improved Offset-Analysis Using Multiple Timing-References

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 an extension to existing approaches that capture and exploit timing-correlation between tasks for scheduling analysis in distributed systems. Previous approaches consider a unique timing-reference for each set of time-correlated tasks and thus, do not capture the complete timing-correlation between task activations. Our approach is to consider multiple timing-references which allows us to capture more information about the timing-correlation between tasks. We also present an algorithm that exploits the captured information to calculate tighter bounds for the worst-case response time analysis under a static priority preemptive scheduler.

1. Introduction

With increasing embedded system complexity, timing-correlation between task activations and communication timing grows. However, even if such correlations can have a large influence on system timing, as shown in [8] [4], most formal scheduling analysis ignore them to avoid the growing analysis complexity.

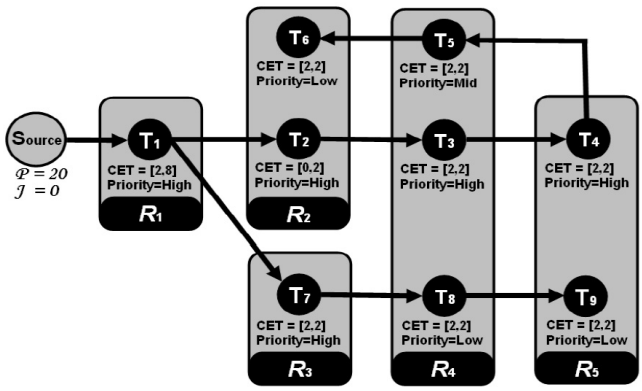

Figure 1. Distributed system example

Take a look on the system in Figure 1. The system is composed of nine tasks mapped on five resources. Let us focus on the worst-case response time calculation on the resource R_4. Due to the data-dependency between the tasks T_3, T_5 and T_8, their activating events are time-correlated, which could rule out simultaneous activation of all three tasks. We call the information about such correlation *inter event stream context* [2]. However, a typical scheduling analysis would ignore the available inter event stream context information and would assume that all three tasks are independent and that in the worst-case they are activated simultaneously [9]. This may result in pessimistic calculated worst-case response times.

Methods exploiting inter event stream contexts for the scheduling analysis already exist [8, 4, 3]. However, they are unable to capture the complete timing-correlation between tasks, because they would consider the external events produced by *Source* as unique reference to capture the timing-correlation between T_3, T_5 and T_8.

In [1], we presented a new technique allowing capturing more information about the timing-correlation between tasks in parallel paths by considering the completion time of the task at which the parallel paths start, as additional reference: e.g. the completion time of T_1 is considered as additional reference to capture the timing-correlation between the activations of T_3 and T_8 and the activations of T_5 and T_8. Even though improvements can be obtained using this technique, it does not allow to capture the complete timing-correlation between tasks belonging to a same single path, e.g. the timing-correlation between activations of T_3 and T_5.

In this paper, we extend the technique presented in [1] by considering additional timing-references in the system to capture the timing-correlation between tasks belonging to a single path or to parallel paths in a more accurate way.

The paper is organized as follows: in the next section, we review the existing approaches from literature that exploit inter event stream contexts for scheduling analysis. In Section 3, we present our computational model before giving an overview about inter event stream context preliminaries in Section 4. In Section 5, we show the limits of existing techniques and extend the relative offset and relative jitter approach. An algorithm exploiting the relative offset and jitter information for the worst-case response time calculation is presented in Section 6. In Section 7 we carry out experiments and interpret the results, before we draw our conclusions.

2. Related Work

Tindell developed in [8] a technique to capture the timing-correlation between tasks in a way that can be exploited by scheduling analysis. Tindell introduced this idea for tasks scheduled under a static priority preemptive scheduler. 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 an external event at the transaction input. An activation of a task releases the execution of one instance of that task, called *job*. A limitation of Tindell's technique is that offsets are not allowed to be larger than the transaction period.

Tindell's work was later generalized by Palencia and Harbour [4]. They presented the WCDO-algorithm (Worst Case Dynamic Offsets), which allows task offsets to be larger than the transaction period. In [6], they also presented an algorithm that exploits the inter event stream context information for tasks scheduled under EDF scheduling. In [5], they presented a new analysis technique exploiting the precedence relation between tasks scheduled under a static priority preemptive scheduler. However, the presented algorithm, called WCDOPS (Worst Case Dynamic Offsets with Priority Schemes), only took into account tasks in linear transactions, where each task is allowed to have at most one output.

Recently, Redell extended the WCDOPS algorithm by considering the precedence relation between tasks in so called tree-shaped transactions [3], i.e. tasks are allowed to have more than one output. Even though the proposed algorithm (the WCDOPS+ algorithm) allows to exploit the precedence relation in a more accurate way, it was based on the inter event stream context capturing technique presented by Tindell, which only considers the external events at the transaction input as unique reference to calculate offsets.

In [1], we presented a new approach that allows to capture the timing-correlation between tasks belonging to parallel paths in a more accurate way by considering the completion time of tasks having several outputs as additional timing-references. Then, so called *relative offset and jitter* are calculated for each task belonging to a path starting at an output of the multi-output task. We also developed an algorithm to exploit the relative offset and jitter information for the worst-case response time calculation under a static priority preemptive scheduler.

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. The *core execution time* of a task, i.e. execution assuming no interrupts, is noted CET. After finishing its execution, a task produces simultaneously 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*.

We assume static priority preemptive scheduling on the resources, i.e. tasks are assigned priorities and the execution of a lower priority task can be interrupted by the execution of higher priority tasks 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. Preliminaries

In this section, we review preliminaries about the offset analysis, as presented in [8] and [4]. Each set of time-correlated tasks in the distributed system is grouped into one transaction: e.g. in the system in Figure 1 all nine tasks belong to a same transaction, which is activated by the events produced by *Source*. In addition, each task belonging to a transaction is identified by an offset parameter which indicates the earliest activation instant of the task after the arrival of the associated external event activating the transaction: e.g. the offset of T_1 is $\Phi_1 = 0$, since T_1 is activated immediately after the arrival of an event at the transaction input. 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 transactions to its *busy window*. The busy window 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 the same resource with T_l. The instant that starts the busy window is called *critical instant* and is noted t_c.

Let now $hp_\Gamma(T_l)$ be the set of tasks belonging a given transaction Γ and to $hp(T_l)$. In [8] and [4], it was shown that the maximum contribution of Γ to the busy window is obtained when the critical instant t_c coincides with the activation time of some task $T_c \in hp_\Gamma(T_l)$, when T_c is delayed by its maximum jitter. We say that T_c determines the critical instant t_c. The maximum contribution of each other task T_i from $hp_\Gamma(T_l)$ to the busy window is obtained by activating each of its jobs, if possible, at or as soon as possible after t_c.

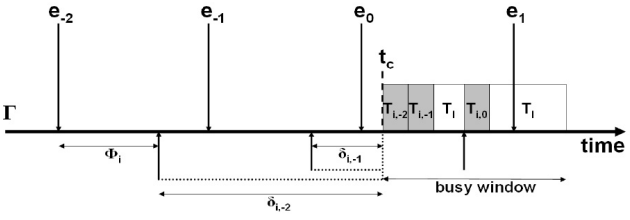

Figure 2. transaction Γ with executions of T_l and jobs from T_i

This is shown in Figure 2. The downward arrows indicate the external events activating the transaction. The upward arrows indicate the offset Φ_i of T_i. Each job of T_i is delayed,

if possible, by a certain amount of time denoted δ_i, to occur at t_c. Note that $\delta_i \leq J_i$. Jobs of T_i, whose activations always occur after t_c, are activated without any delay. Note that jobs of T_i that cannot be activated at t_c, even with a maximum delay, do not contribute to the busy window.

5. Relative Offset And Relative Jitter

5.1 Problem Formulation

Observe the system in Figure 1. We assume static priority scheduling on all resources. Priorities and core execution times of tasks are shown in the figure. T_1 is assumed to be activated periodically by events sent by the source task *Source*. Let the activating event model at the input of T_1 be $(P_1 = 20, J_1 = 0)$. Due to the data-dependency between all tasks, they are grouped into one transaction Γ_{Source}. 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 calculation of the task T_8 on R_4. We use the compositional performance analysis methodology to perform the response time calculation [7], i.e. initially, all external event models are propagated along all system paths until an event model is available for each task, then local scheduling analysis and event model propagation are performed alternately. This process is repeated until no newly propagated event model is different from the event model that was propagated in the previous step, or if some timing constraint is violated. Table 1 shows the activating event models and the global offsets at the inputs of T_3, T_5 and T_8 after having analyzed the resources R_1, R_2, R_3 and R_5. Note that the jitter is due to the response time variation of T_1 and T_2.

task	input event model	global offset
T_3	$(P_3 = 20, J_3 = 8)$	$\Phi_3 = 2$
T_5	$(P_5 = 20, J_5 = 8)$	$\Phi_5 = 6$
T_8	$(P_8 = 20, J_8 = 6)$	$\Phi_8 = 4$

Table 1. Input event models and global offsets at the inputs of T_3, T_5 and T_8

Now having the activating event model of all tasks mapped on R_4, we can calculate the worst-case response time of the lower priority task T_8. The worst-case response time of T_8 calculated by the WCDO [4], WDOPS [5] and WCDOPS+ [3] algorithms is $R_8^w = 6$. This worst-case response time calculation using the WCDO algorithm is shown in the gantt-chart in Figure 3. The worst-case scenario is obtained when T_3 determines the critical instant, i.e. T_3 is activated after having arrived as late as possible. The maximum contribution of T_5 and T_8 to the busy-window is obtained by activating them as soon as possible after the critical instant, i.e. by activating them with a delay, which causes them to coincide with the critical instant. Therefore, two interrupts of T_8 by T_3 and T_5 are calculated.

When using the technique presented [1] for the calculation, the events produced at the outputs of T_1 are considered as additional reference to capture the timing correlation between the tasks mapped on R_4. Exploiting this additional

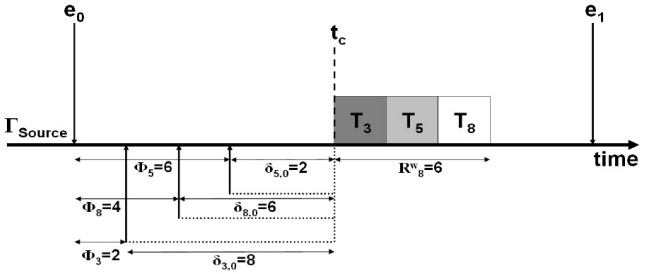

Figure 3. W-C response time calculation of T_8 using the WCDO algorithm

timing-reference results in a different calculated activation instant of T_5: $t_c + 2$. However, the obtained worst-case response time of T_8 is the same as the one calculated using the WCDO, WDOPS and WCDOPS+ algorithms.

Now, let us take a closer look on our system. Since T_4 has the highest priority on R_5, its execution takes 2 time units in both best- and worst-case to finish. Since an execution of T_4 activates T_5, a gap of 2 time units always exists between an execution of T_3 and an execution of T_5. Since this gap is large enough to allow T_8 to finish its execution, an execution of T_8 can either be interrupted by an execution of T_3 or by an execution of T_5, but not by both. Therefore, the true worst-case response time of T_8 is $R_8^w = 4$.

So why are the previous calculations pessimistic? The reason is that the WCDO, WDOPS and WCDOPS+ algorithms consider the arrival of external events at the system inputs (here events produced by *Source*) as unique reference to capture the timing correlation between tasks in a transaction. The algorithm presented in [1] on his part, considers an additional timing-reference (here events produced by T_1), however this reference allows to capture more timing-correlation between tasks belonging to parallel paths but not between tasks belonging to a same single path (here T_3 and T_5). Therefore, not all correlations between tasks activations are captured. E.g. the algorithms cannot recognize that if T_3 is activated after having experienced a maximum delay, the activation of T_5 will experience at the minimum the same delay, since T_3 and T_5 belong to a same path and T_3 precedes T_5. Such correlations could be captured by taking other timing references into account. In our example, it would be more accurate to consider the events arrival at the input of T_3 as additional reference to capture the timing correlation between the activations of T_3 and T_5.

5.2 Relative Offset and Relative Jitter Concept

To capture the timing-correlation between tasks in distributed systems in a more accurate way, we extend the concept of *relative offset and relative jitter*, which we already introduced in [1].

Definition 1 (relative offset) *A task T_i is said to be activated after an offset $\Phi_i(ref)$ relative to a timing-reference ref, if T_i is activated at the earliest, when a relative time $\Phi_i(ref)$ elapses after the occurrence of the timing-reference*

ref. $\Phi_i(ref)$ *is called offset of* T_i *relative to the timing reference ref*.

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

A timing-reference could be the events arrival at the inputs of a task T_r. In this case, the relative offset and jitter information of T_i is denoted: $(T_r, in\ ,\ \Phi_i(T_r, in)\ ,\ J_i(T_r, in))$. If the timing-reference corresponds to the completion time of a task T_r, the relative offset and jitter information of T_i is denoted: $(T_r, out\ ,\ \Phi_i(T_r, out)\ ,\ J_i(T_r, out))$.

In the example in Figure 1, since T_5 is activated exactly 4 time units after the activation of T_3, the offset and jitter information of T_5 relative to the events arrival at the input of T_3 is: $(T_3, in\ ,\ 4\ ,\ 0)$. The offset and jitter information of T_3 relative to the events arrival at its input is: $(T_3, in\ ,\ 0\ ,\ 0)$.

6. Worst-Case Response Time Calculation

In this section, we present an algorithm that exploits the relative offset and jitter information for the worst-case response time calculation. First, we perform the calculation only for tasks belonging to a same single path. Then, we recall the algorithm presented in [1], which performs the calculation for tasks belonging to parallel paths. Finally, we merge both algorithms to obtain a single algorithm that exploits the relative offset and jitter information for distributed systems with tree-shaped task-dependencies.

6.1 Tasks Belonging to a Single Path

We begin deriving the worst-case response time calculation for a lower priority task T_l considering global offsets only. Then we will additionally consider the relative offset and jitter information.

6.1.1 Global offsets exploitation

As already introduced in Section 4, to calculate the maximum contribution of a transaction Γ to the worst-case response time of T_l we begin constructing a critical instant t_c that starts the worst-case busy window. Let us assume that t_c is determined by some task $T_c \in hp_\Gamma(T_l)$. 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 have negative indexes assigned. Subsequent external events have positive indexes assigned. For each task T_i from $hp_\Gamma(T_l)$, 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 $t_{i,k}$. Depending on the critical instant t_c, the global offset and the jitter, each job is categorized into one of the following sets:

- *Set*0: Jobs whose activations cannot occur inside the busy window even with a maximum delay.
- *Set*1: Jobs whose activations can be delayed to coincide with t_c.
- *Set*2: Jobs whose activations always occur after t_c.

In [8] and [4], it was proven that the maximum contribution of a task T_i to the busy window is obtained by delaying the activations of its jobs belonging to *Set*1 by a certain amount of jitter to coincide with t_c and activating its jobs belonging to *Set*2 as soon as possible, i.e. without any delay. E.g. in Figure 2, the maximum contribution of T_i to the busy window 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 from *Set*0 are not considered since they cannot be delayed enough to be activated at t_c.

6.1.2 Relative offsets and jitter exploitation

So far, to calculate the maximum contribution of a job $T_{i,k}$ to the worst-case busy window of T_l, we determined its activation instant depending on its global offset only. However, 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: when considering the global offsets only, we found out that the activations of the jobs $T_{3,0}$ and $T_{5,0}$ can be both delayed to coincide with t_c. However, when considering the available offset and jitter information relative to the events arrival at the input of T_3, we found out that the activation of $T_{3,0}$ always precedes the activation of $T_{5,0}$, by exactly 4 time units and thus, it is impossible to activate both jobs simultaneously at t_c.

In the following, we show the maximum contribution of tasks belonging to $hp_\Gamma(T_l)$ to the worst-case busy window of T_l, when considering both global offsets and relative offset and jitter information.

Let us denote the activation scenario of jobs triggered by the external event e_k by A_k. When a maximum contribution to the busy window is assumed for a job with index k (i.e. the job is activated at t_c), we say that this job determines the activation scenario A_k. Let us now assume that A_k is determined by a job $T_{i,k}$, which belongs to *Set*1. Let T_j be a task from $hp_\Gamma(T_l)$ that belongs to the same path than T_i. In order to calculate the contribution of $T_{j,k}$ to the busy window, we categorize T_j into one of the following sets:

- *Predecessors*(T_i): Set of tasks belonging to $hp_\Gamma(T_l)$ and *preceding* T_i.
- *Successors*(T_i): Set of tasks belonging to $hp_\Gamma(T_l)$ and *preceded* by T_i.

Once the task T_j have been categorized into the sets above, the activation instant of $T_{j,k}$ that leads to its maximum contribution to the busy window can be determined according to the following theorem:

Theorem 1 *Assuming a maximum contribution of a job* $T_{i,k}$ *from Set*1 *to the busy window by activating it at* t_c*, a job* $T_{j,k}$ *can only contribute to the busy window if* T_j *belongs to Successors*(T_i). *The maximum contribution of* $T_{j,k}$ *is obtained by activating it at the instant:* $t_{j,k} = t_c + \Phi_j(T_i, in)$,

where $\Phi_j(T_i, in)$ is the offset relative to the event arrival at the input of T_i.

Proof 1 *Let us assume that T_j belongs to Predecessors(T_i). In this case, $T_{i,k}$ is activated after $T_{j,k}$ finishes its execution. Therefore $T_{j,k}$ is executed before t_c, and thus it does not contribute to the busy window. We assume now that T_j belongs to Successors(T_i). In this case $T_{j,k}$ is activated after $T_{i,k}$ finishes its execution. Therefore $T_{j,k}$ is activated after t_c, and thus it may contribute to the busy window. The maximum contribution of $T_{j,k}$ to the busy window is obtained by activating it as soon as possible after t_c. Since $T_{i,k}$ is activated at t_c, the earliest activation instant $t_{j,k}$ of $T_{j,k}$ occurs when $T_{j,k}$ is activated without any delay relative to the event arrival at the input of T_i. Therefore, the activation instant that leads to a maximum contribution of $T_{j,k}$ to the busy window is $t_{j,k} = t_c + \Phi_j(T_i, in)$.*

Let us apply theorem 1 to the system example in Figure 1. Assuming that $T_{5,0}$ determines the activation scenario A_0, $T_{3,0}$ would not contribute to the busy window since T_3 belongs to *Predecessors*(T_5). Assuming now that the job $T_{3,0}$ determines the activation scenario A_0, the activation instant $t_{5,0}$ that could lead to a maximum contribution of $T_{5,0}$ to the busy window is according to theorem 1: $t_{5,0} = t_c + \Phi_5(T_3, in) = t_c + 4$ (in this case, since T_8 finishes its execution at the instant $t_c + 4$, the activation of $T_{5,0}$ occurs outside the busy window).

Given a job $T_{i,k}$ that belongs *Set*1 and determines the activation scenario A_k, we can now calculate the maximum contribution to the busy window for all jobs involved in A_k and belonging to tasks situated on the same path than T_i. This is given by the following algorithm:

Algorithm 1 MSC: Maximum Scenario Contribution

Input: busy window, $T_{i,k}$
Output: maxContribution

```
1: maxContribution = 0;
2: for all task T_j belonging to hp_Γ(T_l) do
3:    if T_j = T_i or T_j belongs to Successors(T_i) then
4:       t_{j,k} = t_c + Φ_j(T_i, in);
5:       if t_{j,k} is within busy window then
6:          maxContribution = maxContribution + CET_j^max;
```

Note that if all jobs involved in a given activation scenario A_k belong to *Set*2, none of these jobs can be activated at t_c. Therefore, no job can determine the activation scenario A_k. In this case, the maximum contribution of these jobs to the busy window is obtained when they are activated, as before (see Section 6.1.1), without experiencing any delay.

6.2 Tasks Belonging to Single or Parallel Paths

As mentioned in Section 2, we already presented an approach to capture the relative offset and jitter information between tasks in parallel paths [1]. The main idea of this approach is to consider the completion time of each task having several outputs as additional timing-reference. Consider the resource R_5 in Figure 1. The offset and jitter information relative to the completion time of T_1 can be exploited when calculating the contribution to the busy window of activation scenarios, where jobs from T_4 and T_9 are involved, since this reference is the *most recent* common reference between the two tasks. In the following, we recall how to exploit the relative offset and jitter information for tasks belonging to parallel paths (Note that theorem 1 cannot be applied for such tasks since there is no precedence relation between them). Proofs and details can be found in [1].

Let T_i be a task from $hp_\Gamma(T_l)$ that belongs to a path starting at an output of some task T_r. Let us assume that the job $T_{i,k}$ from *Set*1 determines the activation scenario A_k, i.e. $T_{i,k}$ is activated at the critical instant t_c. Let $\delta_{i,k}$ be the delay needed by $T_{i,k}$ to be activated at t_c (see $\delta_{i,k}$ in Figure 2). Let now T_j be another task from $hp_\Gamma(T_l)$ that belongs to a path starting at another output of T_r. In [1] we proved that the activation instant $t_{j,k}$ of $T_{j,k}$ occurs within the interval $[t_{j,k}^{min}, t_{j,k}^{max}]$, where:

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

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

Depending on whether t_c occurs within the interval $[t_{j,k}^{min}, t_{j,k}^{max}]$, it was shown that the activation instant $t_{j,k}$ that may lead to a maximum contribution to the busy window occurs:

- before t_c, if $t_{j,k}^{max} < t_c$ (i.e no contribution the busy window)
- at $max(t_c, t_{j,k-1})$, if $t_c \in [t_{j,k}^{min}, t_{j,k}^{max}]$
- at $t_{j,k}^{min}$, if $t_c < t_{j,k}^{min}$

Now, we can extend the algorithm MSC to make it consider tasks belonging to a single path as well as tasks belonging to parallel paths.

Algorithm 2 MSC: Maximum Scenario Contribution

Input: busy window, $T_{i,k}$
Output: maxContribution

```
1: maxContribution = 0;
2: for all task T_j belonging to hp_Γ(T_l) do
3:    if T_i and T_j belong to a same path then
4:       if T_j = T_i or T_j belongs to Successors(T_i) then
5:          t_{j,k} = t_c + Φ_j(T_i, in);
6:    else if T_i and T_j belong to parallel paths then
7:       calculate [t_{j,k}^min, t_{j,k}^max];
8:       determine t_{j,k} depending on t_c and [t_{j,k}^min, t_{j,k}^max];
9:    if t_{j,k} is within busy window then
10:      maxContribution = maxContribution + CET_j^max;
```

Depending on whether $T_{j,k}$ belongs to a same single path than $T_{i,k}$ or not, the new MSC algorithm calculates the activation instant of $T_{j,k}$ either by exploiting the offset and jitter information relative the event arrival at the input of T_i (line $3-5$) or by exploiting the offset and jitter information relative the completion time of the task at which start the parallel paths to which T_i and T_j belong (line $6-8$). Let us apply this for the response time calculation of T_8 on the resource R_4. Assume that the job $T_{3,0}$ determines the activation scenario A_0. The activation instant of $T_{8,0}$ is calculated by exploiting the offset and jitter information relative to the completion time of T_1, since T_3 and T_8 belong to parallel paths starting at the outputs of T_1. When calculating the activation instant of $T_{5,0}$, it is more accurate to exploit the offset and jitter information relative to the arrival of events at the input of T_3, since T_3 and T_5 belong to a same path.

7. Experiments

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

Additionaly, we show the results obtained using the classical WCDOPS+ algorithm, which only considers external events as timing references when exploiting the precedence relation between tasks, compared with the results obtained using a modified WCDOPS+ algorithm which takes into account relative offset and jitter (due to space restriction, the modified WCDOPS+ algorithm cannot not be presented in this paper).

We also compared the results obtained using the classical WCDOPS+ algorithm, which only considers external events as timing references when exploiting the precedence relation between tasks, with the results obtained using a modified WCDOPS+ algorithm which takes into account relative offset and jitter (due to space restriction, the modified WCDOPS+ algorithm cannot not be presented in this paper).

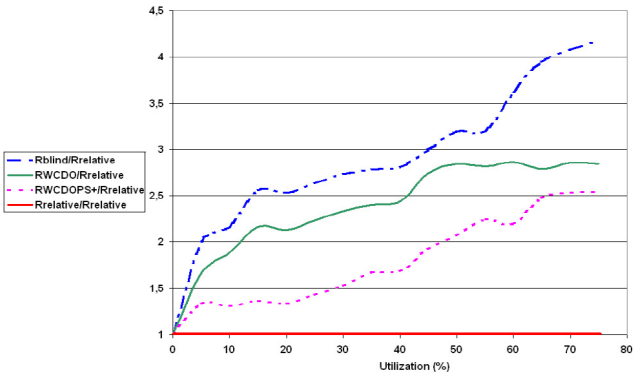

Figure 4. Response time average ratios

Figure 4 shows the response time average ratios as function of the system utilization. The results show that a large improvement can be obtained when considering multiple timing-references: up to 75% compared to the inter event stream context blind analysis, up to 64% compared to the WCDO and up to 60% compared to the classical WCDOPS+. It is also interesting to note that 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 multiple timing-references, the effect of global jitters on the response time calculation can be reduced and thus, lower response-times are calculated.

8. Conclusion

In this paper we extended existing approaches capturing timing-correlation between tasks in distributed systems by considering multiple timing-references. We have seen that considering the external events at the system inputs as unique timing-reference could lead to pessimistic response times calculation. Our solution consists in considering the most recent timing-reference for each task to perform an exacter calculation of its activation instant. We also developed an algorithm that exploits this approach for the worst-case static priority preemptive analysis. Our experiments show that our technique allows to calculate considerably tighter bounds compared to previous 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] R. Henia and R. Ernst. Context-aware scheduling analysis of distributed systems with tree-shaped task-dependencies. In *Proc. of Design, Automation and Test in Europe (DATE'05)*, Munich, Germany, Mar. 2005.

[2] 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.

[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] J. C. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Proc of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2003.

[7] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proc. of Design, Automation and Test in Europe (DATE'02)*, Paris, France, Mar. 2002.

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

[9] 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.