Scenario Aware Analysis for Complex Event Models and Distributed Systems

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

Abstract

The set of executing tasks in modern hard real time systems may change during system execution. This change, called scenario change, may lead to a transient overload situation due to the interference of different task set executions, thus necessitating timing requirement verification. Previously developed approaches analyzing response times across scenario changes are limited to strict periodic task event models and restricted to uniprocessor systems, while existing methods adapted for the analysis of distributed systems are not suitable for the analysis across scenario changes. In this paper, we eliminate the restrictions concerning task event models and present a scheduling analysis methodology allowing response time calculation across a scenario change for multi-scenario distributed systems.

I. Introduction

Modern hard real-time systems often run in different operating modes, also called scenarios. Each scenario is characterized by a different behavior and is associated with a specific set of tasks. During a change from an old scenario to a new scenario, some tasks may be removed while others may be released for the first time. This may lead to a transitional phase during which the set of executed tasks may include tasks belonging to both scenarios. For instance, to preserve data-consistency, some old scenario tasks may be allowed to terminate their execution after the scenario change, thus interfering with the execution of new scenario tasks and leading to a transient overload situation. Therefore, it is necessary to verify the timing requirements across a scenario change to ensure that no deadlines are missed.

Numerous formal scheduling analysis approaches for uniprocessor [7] [14] and distributed systems [3] [13] [9] have been developed by the real-time system research community for decades. However, these techniques are only applicable to systems with a static task set or multi-scenario systems with mutual exclusive scenario executions, i.e. execution interference of old and new scenario task sets is excluded.

Methods analyzing the timing behavior across a scenario change under priority preemptive scheduling already exist [11] [16] [8]. However, they are limited to purely periodic task activation patterns. In addition, they exclude interference of successive task executions with each other, by restricting the task deadline to be smaller than the task period. Furthermore, these approaches are limited to uniprocessor systems.

In this work, we present an approach for the response time analysis across a scenario change under static priority preemptive scheduling. We eliminate the restriction concerning task deadlines and also allow complex task activation patterns including periodic with jitter, periodic with burst and sporadic event models. This step is essential for the second part of the paper, where we present an approach for the response time analysis across a scenario change for distributed systems.

The rest of the paper is structured as follows. In the next section, we review the existing scheduling analysis approaches from the literature. In Section III-A, we introduce our application model. Task activating event models are presented in detail in Section III-B. In Section IV, we describe the scenario change protocol we use in this work. In Section V, we present a scheduling analysis approach allowing response time calculation across a scenario change for tasks with complex activation patterns in uniprocessor systems. Based on this, we present an analysis method allowing response time calculation across a scenario change in multi-scenario distributed systems in Section VI.

II. Related Work

The goal of the scheduling analysis is to verify that a system implementation meets all response time and throughput constraints, e.g. end-to-end deadlines. Numerous scheduling analysis techniques have been developed by the real-time system research community for decades. The first schedulability analysis algorithm was presented by Liu and Layland for Rate Monotonic Scheduling [7]. Since then, various scheduling analysis algorithms have been developed. Examples include Static Priority Preemptive [15] and time-slicing mechanisms like TDMA or Round-Robin [5]. Scheduling analysis algorithms are not directly applicable to distributed systems. From the literature, two formal approaches addressing this problem can be identified: the holistic and the compositional approach. The holistic approach extends the classical scheduling theory to distributed systems [14] [3], while the compositional approach couples local scheduling analysis techniques on individual components via event streams [2] [13] [9]: the output event stream of one component turns into the input event stream of the connected component. System performance analysis can then be performed by iteratively alternating local scheduling analysis and event stream propagation between components. However, common to these methods is that they are not applicable to multi-scenario distributed systems, i.e. the executed task set in the system is assumed to be static over time.

Response time calculation across scenario changes have been addressed in [11] [16] [8] (note that we are using the term "scenario" instead of "mode" in order to make a distinction with the input dependent task behavior also called "mode" [17]). In [11], Sha et al present a schedulability test across a scenario change according to the rate monotonic scheduling policy, where tasks can either be added or removed depending on the system utilization. Tindell et al. showed in [16] that the analysis in [11] is not sufficient, since the test may pass a task set that is unschedulable. Tindell et al. propose a worst-case response time analysis algorithm across a scenario change under deadline monotonic scheduling. The scenario change protocol they use defines three task types: old scenario tasks that are allowed to complete their executions after a scenario change, unchanged tasks that execute independently from the scenario change and new scenario tasks that are either introduced for the first time after the scenario change or that represent a modified versions of old scenario tasks. The analysis algorithm they propose is however limited to strict periodic activating event model. In addition, task deadlines are restricted to be smaller than the task period. Furthermore, the algorithm is limited to uniprocessor systems.

In this paper, we present a new analysis technique for the response time calculation across a scenario change under static priority preemptive scheduling that eliminates the restrictions in [16] regarding task activation patterns and deadlines. Then, we extend our approach to calculate response times across a scenario change for multi-scenario distributed systems.

III. System Model

In this section, first we present our application model. Then, we give an overview about event models.

A. Application Model

An application is modeled by a set of computation and communication tasks (application entities). The tasks are mapped and executed on a set of processing (CPUs) and communication (Buses) elements, representing the system architecture. Each task T_i is characterized by its *core execution time* interval $[c_i^{min}, c_i^{max}]$, defined as the minimum and maximum times T_i requires for a complete execution on the corresponding resource, assuming that no blocking or preemption occur during execution.

A task graph describes the functional and timing dependencies between tasks. Tasks are allowed to have more than one immediate successor and predecessor. The task graph may contain cycles, describing applications with cyclic dependencies between tasks, like control loops [4].

One execution of a task is called job. $T_{i,k}$ denotes the k-th job of task T_i, assuming that the time instance at which $T_{i,1}$ occurred is known. The activation of a task is triggered by an activating event. Activating events can be generated in a multitude of ways, including expiration of a timer, external or internal interrupt, and task chaining. Each task is assumed to have one input FIFO. A task reads its activating event from its input FIFO and writes data into the input FIFO of a dependent task. A task may read its input data at any time during one execution. We therefore assume that the data needs to be available at the input during the whole execution of the task. We also assume that input data is removed from the input FIFO at the end of the task execution. These assumptions are standard in scheduling analysis. After finishing its execution, a task produces one event at each of its outputs.

All activating events of a task are captured by an event stream. The behavior of an event stream is described using parameterized event models, presented in detail in Section III-B.

As already stated, a task needs to be mapped on a *computation* or *communication resource* to execute. When multiple tasks share the same resource, then two or more tasks may request the resource at the same time. In order to arbitrate request conflicts, a resource is associated with a *scheduler* which selects a task to be executed out of the set of active tasks according to some scheduling policy. For each task T_i, *scheduling analysis* calculates worst-case (r_i^{max}) and also best-case (r_i^{min}) task response times, i.e. the maximum and minimum times between task activation and task completion. Scheduling analysis guarantees that all observable response times of T_i will fall into the calculated $[r_i^{min}, r_i^{max}]$ interval. We therefore say that scheduling analysis is *conservative*. In this paper we use $r_{i,k}$ to refer to the response time of the k-th job of task T_i.

B. Event Models

The timing of the events within an event stream is described using *event models*. For our model we use a six-class parameterized event model set that slightly extends the known event models from literature to capture the consequences of jitters, and subsequently bursts [9]. Each event model is described using a set of three parameters: *period* (p), *jitter* (j) and *minimum distance* (d^{min}). Depending on the event model, only a part of the three parameters must be defined: a periodic stream requires only the period to be specified, a periodic stream with jitter requires two parameters, the period and the jitter, while a periodic stream with burst is specified using all parameters, period, jitter and minimum distance.

The period shows the rate at which a task is activated and is used to compute the average load determined by the task on the assigned resource. A lot of applications use periodic event models to specify the activation pattern of the tasks. Examples can be find in signal processing domain or control engineering.

The jitter parameter is used to show the perturbations that may influence the pure periodic activation of a task. A periodic with jitter event model describes the activation of a task that still has a general periodic behavior, but the single activations may occur within a time interval, rather than at exact time instances. The maximum value of this time interval is expressed using the jitter parameter. In general, if a jitter is present at task input then it is propagated to the output. Moreover, a variable task execution time and the effects of scheduling induce additional jitter at the output. In this paper we use j_i to denote the jitter parameter of the activating event models of task T_i.

When the jitter interval exceeds the period, two or more events may arrive at the same time, leading to bursty task activations. The minimum distance parameter d^{min} is used to additionally specify the timing of the activating events in case of bursts, and to reduce the transient load peaks.

Mathematically, the event models are represented using four functions: $\eta^+(\Delta t)$, $\eta^-(\Delta t)$, $\delta^-(n)$ and $\delta^+(n)$ [9].

Definition 1 (Upper-Bound Arrival Function) *The* upper bound *arrival function* $\eta^+(\Delta t)$ *specifies the* maximum *number of events that can occur during any time interval of length* Δt.

Definition 2 (Lower-Bound Arrival Function) *The* lower bound *arrival function* $\eta^-(\Delta t)$ *specifies the* minimum *number of events that have to occur during any time interval of length* Δt.

Definition 3 (Minimum Distance Function) *The* minimum *distance function* $\delta^-(n)$ *specifies the* minimum *distance between any event x and the (n-1)-th* $(n \geq 2)$ *event after x in the corresponding event stream.* $\delta^-(1)$ *is set to* 0.

Definition 4 (Maximum Distance Function) *The* maximum *distance function* $\delta^+(n)$ *specifies the* maximum *distance between any event x and the (n-1)-th* $(n \geq 2)$ *event after x in the corresponding event stream.* $\delta^+(1)$ *is set to* 0.

The mathematical representation of the arrival and the distance functions is given in the appendix. In the next sections we use the four functions to capture the task workload in the response time equations.

IV. Scenario Change Protocol

In this section, we describe the system behavior across a *scenario change* (*SC*). A scenario change is triggered by a *scenario change request* (*SCR*). A *SCR* is produced either due to a change in the system environment or due to a system transition to a specific internal state requiring a scenario change. The instant at which a *SCR* occurs is denoted by t_{SCR}. To avoid the interference of several scenario changes, a *SCR* is assumed to occur at any instant during a scenario execution, but never during another *SC*.

Consider a *SC* from an old to a new scenario. Depending on the task behavior across the scenario change, we define three task types:

- *unchanged* task: an unchanged task belongs to both scenarios task sets. It remains unchanged and continues executing normally after the *SCR*.
- *completed* task: a completed task only belongs to the old scenario task set. However, to avoid the lose of data consistency, completed task jobs activated before t_{SCR} are allowed to complete their execution after the *SCR*. After that, the task terminates.
- *added* task: an added task only belongs to the new scenario task set. It is activated for the first time after the *SCR*. Each added task is assigned an *offset* value ϕ that denotes its earliest activation time after t_{SCR}.

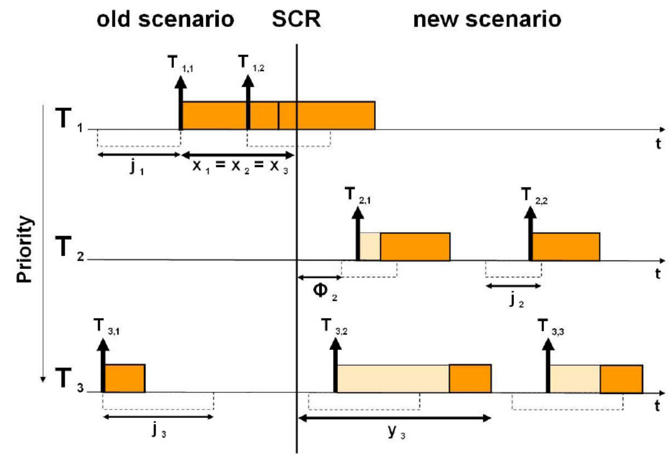

Fig. 1 - SCHEDULING EXAMPLE DURING A SCENARIO CHANGE

The different task behaviors across a *SC* are shown in Figure 1. The figure shows a scheduling example of a set of three tasks mapped on a static priority preemptive scheduled resource. The tasks are activated periodically with jitter. The jitter intervals are represented with dashed lines. The up-pointing arrows correspond to the task activations within the jitter intervals. Dark-colored boxes represent task executions, while light-colored boxes represent the time a task is suspended waiting for execution. Task T_1, which is a completed task, is allowed to finish executing the job $T_{1,2}$, which has been activated earlier than the *SCR*. Task T_2, which is an added task, is activated for the first time in the new scenario after the *SCR* occurs. Task T_3, which is an unchanged task, executes in both scenarios regardless of the *SC*. Note that the task classification depends on the scenarios involved in the scenario change, e.g. T_3, which is an unchanged task in the *SC* illustrated in Figure 1, could be a completed task in a *SC* involving other scenarios.

During a *SC* there could be a transition phase during which tasks from both scenarios are executing. This could lead to a transient overload on the resource. In the scheduling example in Figure 1, the transition phase starts at the *SCR* and ends when $T_{3,2}$ finishes executing. During this phase completed tasks may delay the execution of added tasks, e.g. $T_{2,1}$ has to wait until $T_{1,2}$ finishes executing. The execution of unchanged tasks may also be delayed and preempted by executions of both completed and added tasks, e.g. the execution $T_{3,2}$ is delayed by the executions of $T_{1,2}$ and $T_{2,1}$. The same is valid for completed tasks. Since the timing requirements in the system have to be met at any time during the system execution, it is necessary to verify if task deadlines could be missed during the transition phase. Thus, we need to perform a worst-case response time analysis during the transition phase of a *SC*.

To show the analysis idea in isolation, first we perform the calculation in Section V for multi-scenario uniprocessor systems. Then, we extend the analysis for multi-scenario distributed systems in Section VI.

V. Analysis For Multi-Scenario Uniprocessor Systems

In this section, we perform the worst-case response time analysis during a *SC* for each task type separately, in a static priority preemptive scheduled uniprocessor system. Tasks are assigned unique priorities. Before we start presenting the analysis, we introduce some definitions and annotations that we use later in the response time calculation.

Consider a task T_i. The set of tasks with higher priorities than T_i is denoted $hp(T_i)$. Additionally, $hp_u(T_i)$, $hp_c(T_i)$ and $hp_a(T_i)$ denote respectively the sets of unchanged, completed and added tasks belonging to $hp(T_i)$.

We also introduce the definition of a *busy window*. A busy window is defined as a time interval in which the resource is busy processing tasks mapped on it. The scheduling example in Figure 1 contains three busy windows: the first busy window corresponds to the execution of the job $T_{3,1}$, the second busy window contains the executions of the jobs $T_{1,1}$, $T_{1,2}$, $T_{2,1}$ and $T_{3,2}$, while the third busy window consists of the execution of the jobs $T_{2,2}$ and $T_{3,3}$. The busy window during which a *SCR* occurs is called *transition busy window*, e.g. the second busy window in Figure 1. The task T_i transition busy window refers to the transition busy window during which the resource is busy processing T_i or tasks from $hp(T_i)$, e.g. the task T_2 transition busy window in the scheduling example in Figure 1 starts at the activation of $T_{1,1}$ and ends when $T_{2,1}$ finishes executing. In the following, the length of the task T_i transition busy window is denoted by w_i. The length of the transition busy window parts before and after t_{SCR} are denoted respectively by x_i and y_i.

For the worst-case response time analysis of T_i we need to calculate the maximum workload of each task from $hp(T_i)$ within w_i. Note that the workload calculation depends on the task type, since completed tasks can only be activated before the *SCR*, added tasks only after the *SCR*, while unchanged tasks can be activated within the whole transition busy window.

We use the iterative approach presented by Lehoczky [6] for the calculation, i.e. we start with w_i equal to zero, calculate the workload produced by the maximum activation number of all tasks during w_i and then repeat the process using the calculated workload as new value for w_i. This process is repeated until w_i converges, i.e. until two consecutive calculated values of w_i are equal or the analyzed T_i misses its deadline.

In the following, for a given transition busy window length w_i, and supposing that we know the instant t_{SCR} at which the *SCR* request occurs, we calculate the maximum workload of each task type. Later in Section V-B we show how to determine t_{SCR}.

A. Maximum Task Workload

Let T_u be a task from $hp_u(T_i)$. Since T_u is an unchanged task, it can be activated at any instant during the transition busy window regardless of the *SCR*. Using the upper-bound arrival function defined in Section III-B, we can compute the maximum activation number of T_u within w_i. The maximum workload of T_u within w_i can therefore be expressed by:

$$\eta_u^+(w_i) \cdot c_u^{max} \tag{1}$$

Let T_c be a task from $hp_c(T_i)$. Since T_c is a completed task, it can only be activated before or at the *SCR*. Therefore, the maximum workload of T_c during w_i corresponds to its maximum workload during x_i and can be expressed by:

$$\eta_c^+(x_i) \cdot c_c^{max} \tag{2}$$

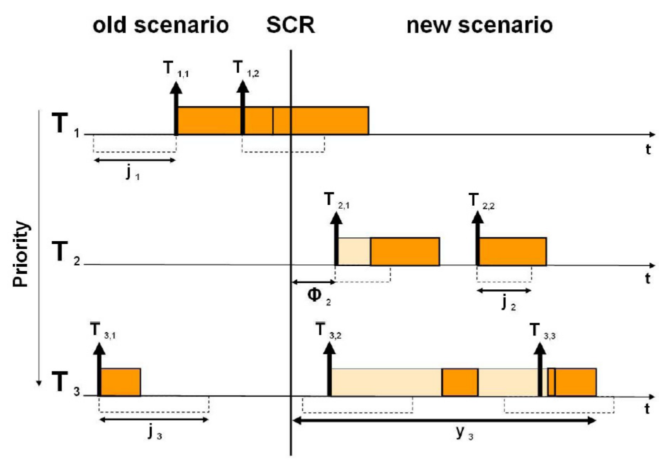

Fig. 2 - MAXIMUM WORKLOAD OF THE ADDED TASK T_2 WITHIN w_3

Let T_a be a task from $hp_a(T_i)$. Since T_a is an added task, it cannot be activated before the instant $t_{SCR} + \phi_a$, i.e. its maximum workload during w_i corresponds to its maximum workload during $y_i - \phi_a$. To ensure a maximum workload of T_a during w_i, its jobs has to be activated as early as possible within their respective jitter intervals. If a job activation occurs namely after the end of the transition busy window, by moving the job activation earlier within its jitter interval, we increase the possibility to activate the job inside the transition busy window. This can be observed by comparing the task T_3 transition busy window lengths in the Figures 1 and 2: in Figure 2, we moved the activations of $T_{2,1}$ and $T_{2,2}$ earlier in the time to occur at the start of their respective jitter intervals, thus making $T_{2,2}$ and also $T_{3,3}$ activated within the transition busy window. Based on this, the maximum workload of T_a during w_i can be expressed by:

$$\left\lceil \frac{y_i - \phi_a}{p_a} \right\rceil_0 \cdot c_a^{max}$$

which is also equal to:

$$\left\lceil \frac{w_i - x_i - \phi_a}{p_a} \right\rceil_0 \cdot c_a^{max} \tag{3}$$

Note that the modified ceiling function used in equation 3 returns zero if $w_i - x_i - \phi_a < 0$.

B. Scenario Change Request

As can be noted in the equations 2 and 3, the maximum workload calculation for completed and added tasks depends on x_i, i.e. it depends on the occurrence of the instant t_{SCR} within the task T_i transition busy window. Let $hp_c^*(T_i)$ be the set of completed tasks with an equal or a higher priority than T_i (i.e. if T_i is a completed task, $hp_c^*(T_i) = hp_c(T_i) \cup \{T_i\}$ otherwise $hp_c^*(T_i) = hp_c(T_i)$). We use the following theorem to determine t_{SCR}.

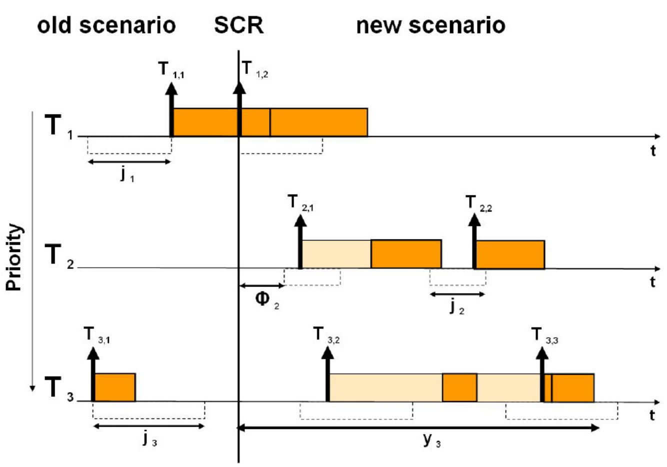

Fig. 3 - t_{SCR} COINCIDES WITH THE ACTIVATION OF THE COMPLETED TASK JOB $T_{1,2}$

Theorem 1 *The worst-case scenario for a task T_i under analysis is obtained, when t_{SCR} coincides with the activation instant of a task from $hp_c^*(T_i)$ within the task T_i transition busy window.*

Proof: Remember that completed tasks can only be activated before or at t_{SCR}, but are allowed to finish their execution after the scenario change. Now suppose that t_{SCR} does not coincide with the activation instant of any completed job within the transition busy window. Let us now move back (i.e. earlier in the time) the instant t_{SCR} until it coincides with the activation of a completed job. By doing this, on the one side we did not exclude the execution of any completed task job from the transition busy window. On the other side, since the whole activation pattern of added tasks was moved earlier in the time together with t_{SCR}, some added task jobs that was activated after the end of the transition busy window, may now be activated inside it. This can be observed by comparing the task T_3 transition busy windows in the Figures 1 and 3: in Figure 3, we moved back t_{SCR} earlier in the time to coincide with the activation of $T_{1,2}$, thus increasing the the tasks T_2 and T_3 workloads since the jobs $T_{2,2}$ and $T_{3,3}$ are now executing within the task T_3 transition busy window. ■

Note that if t_{SCR} coincides with the activation of a given completed job, no statement can be made, if we move ahead t_{SCR} in the time to coincide with the activation of the next completed job within the transition busy window. This is because we added the execution of this completed job to the transition busy window. However, since we moved t_{SCR} and thus the whole activation pattern of added tasks later in the time, we may have excluded the execution of some added task jobs from the transition busy window. Therefore, for worst-case response time analysis of T_i, we need to apply the calculation to all possible values of x_i corresponding to all possible instants t_{SCR} obtained according to the Theorem 1 and choose the value that leads to the worst-case response time of T_i.

Let $\mathcal{X}_i$ be the set of all possible values of x_i determined according to Theorem 1. Algorithm 1 allows to calculate the set $\mathcal{X}_i$:

Algorithmus 1 Calculate $\mathcal{X}_i$

1: calculate $\mathcal{L}_i$
2: **for all** $T_c \in hp_c^*(T_i)$ **do**
3:     calculate $\eta_c^+(\mathcal{L}_i)$
4:     **if** $\eta_c^+(\mathcal{L}_i) \geq 1$ **then**
5:         **for** $n = 1$ to $\eta_c^+(\mathcal{L}_i)$ **do**
6:             add $\delta_c^-(n)$ to $\mathcal{X}_i$
7: remove duplicates from $\mathcal{X}_i$

In the first line, Algorithm 1 calculates $\mathcal{L}_i$, which is the length of the T_i busy window with a maximum workload of unchanged and completed tasks, i.e. the longest busy window within the old scenario execution. The calculation of $\mathcal{L}_i$ is performed by solving the following equation:

$$\mathcal{L}_i^{n+1} = \sum_{\forall T_u \in hp_u(T_i)} \eta_u^+(\mathcal{L}_i^n) \cdot c_u^{max} + \sum_{\forall T_c \in hp^*(T_i)} \eta_c^+(\mathcal{L}_i^n) \cdot c_c^{max} \tag{4}$$

This can be done by using the iterative approach presented earlier in this section. The first clause in equation 4 calculates the maximum workload of unchanged tasks from $hp_u(T_i)$ during $\mathcal{L}_i^n$, while the second clause calculates the maximum workload of completed tasks from $hp_c^*(T_i)$ during $\mathcal{L}_i^n$. The calculation starts with an initial value $\mathcal{L}_i^0 = 0$. For a better understanding of this algorithm step, we apply it to the analysis of task T_2 in the example in Figure 1. The set $hp_c^*(T_2)$ contains the task T_1, while $hp_u(T_2)$ is empty. When starting with $\mathcal{L}_2^0 = 0$, equation 4 calculates $\mathcal{L}_2^1 = c_1^{max}$. This is because at most only one job of T_1 can be activated within a busy window of length 0. In the second iteration, equation 4 calculates $\mathcal{L}_2^2 = 2 \cdot c_1^{max}$. This is because at most 2 jobs of T_1 can be activated during a busy window of length c_1^{max}. In the third iteration, equation 4 calculates $\mathcal{L}_2^3 = 2 \cdot c_1^{max} = \mathcal{L}_2^2$, i.e the calculation converged and $\mathcal{L}_2 = 2 \cdot c_1^{max}$.

In the lines 2 to 6, for each completed task T_c from $hp_c^*(T_i)$, the algorithm 1 calculates all possible values of x_i that could be obtained according to Theorem 1. This is done first by calculating $\eta_c^+(\mathcal{L}_i)$, the maximum activation number of T_c within $\mathcal{L}_i$ (line 3). For each of these activations, the algorithm calculates its corresponding value of x_i, i.e. the minimum distance between the busy window start and the activation occurrence (line 6). This distance can be calculated using the minimum distance function defined in Section III-B. Then, the calculated value is added to $\mathcal{X}_i$. When applying this algorithm part to the analysis of task T_2 in the example in Figure 1, the algorithm calculates $\eta_1^+(\mathcal{L}_i) = 2$, i.e at most 2 jobs of T_1 can be activated during $\mathcal{L}_i$. In the lines 4 to 6, the algorithm calculates for each activation of T_1 its corresponding value of x_2. These values are $\delta_1^-(1) = 0$ and $\delta_1^-(2)$ which corresponds to the distance between the activations of $T_{1,1}$ and $T_{1,2}$ in Figure 1.

Since some completed task jobs may be activated simultaneously leading to equal calculated values of x_i, the algorithm removes in line 7 the duplicates from $\mathcal{X}_i$.

Now having calculated all possible values of x_i, the equations 1, 2 and 3 can be applied for the worst-case response time calculation across a *SC*. In Section V-C, V-D and V-E, the term r_i^{max} refers to the worst-case response time of T_i across a *SC*.

C. Worst-Case Response Time of Unchanged Tasks

Let us assume that T_i is an unchanged task. Algorithm 2 calculates the worst-case response time of T_i across a *SC*.

Algorithmus 2 worst-case response time of an unchanged task

1: $r_i^{max} = 0$
2: **for all** $x_i \in \mathcal{X}_i$ **do**
3: $\quad k = 0$
4: $\quad$ **repeat**
5: $\quad\quad k{+}{+}$
6: $\quad\quad$ calculate $w_{i,k}$ according to equation 5
7: $\quad\quad r_{i,k} = w_{i,k} - \delta_i^-(k)$
8: $\quad\quad r_i^{max} = \max(r_i^{max}, r_{i,k})$
9: $\quad$ **until** $\eta_i^+(w_{i,k}) = k$
10: **return** r_i^{max}

The calculation is performed for each value of x_i from $\mathcal{X}_i$ (line 2). For a given value of x_i, the algorithm computes in each iteration $w_{i,k}$ (line 6), which is the length of the job $T_{i,k}$ transition busy window, i.e. the length of the transition busy window with k executions of T_i and a maximum workload of tasks from $hp(T_i)$. The calculation of $w_{i,k}$ is performed by solving the following equation:

$$
\begin{aligned}
w_{i,k}^{n+1} \quad = \quad & k \cdot c_i^{max} + \\
& \sum_{\forall T_u \in hp_u(T_i)} \eta_u^+(w_{i,k}^n) \cdot c_u^{max} + \\
& \sum_{\forall T_c \in hp_c(T_i)} \eta_c^+(x_i) \cdot c_c^{max} + \\
& \sum_{\forall T_a \in hp_a(T_i)} \left\lceil \frac{w_{i,k}^n - x_i - \phi_a}{p_a} \right\rceil_0 \cdot c_a^{max}
\end{aligned}
\tag{5}
$$

The first clause in equation 5 calculates the workload due to k executions of T_i. The second, third and fourth clauses calculate according to the equations 1, 2 and 3 respectively the maximum workload due to executions of unchanged, completed and added tasks with higher priority than T_i during $w_{i,k}$. The equation starts with a value of $w_{i,k}^0$ equal to $k \cdot c_i^{max}$ and is solved iteratively.

Line 7 of the algorithm computes the response time of the job $T_{i,k}$ within $w_{i,k}$. This is done by subtracting from $w_{i,k}$ the distance between the transition busy window start and the activation instant of $T_{i,k}$. If additional jobs of T_i could be activated within $w_{i,k}$ (line 9), the calculation is repeated for $k = k + 1$ (line 5), otherwise it terminates. The maximum over all $r_{i,k}$ calculated for all values of x_i represents the worst-case response time of T_i across the scenario change (line 8).

D. Worst-Case Response Time of Completed Tasks

Let us assume that T_i is a completed task. Algorithm 3 calculates the worst-case response time of T_i across a scenario change.

Algorithmus 3 worst-case response time of a completed task

1: $r_i^{max} = 0$
2: **for all** $x_i \in \mathcal{X}_i$ **do**
3: $\quad k = 0$
4: $\quad$ **repeat**
5: $\quad\quad k{+}{+}$
6: $\quad\quad$ calculate $w_{i,k}$ according to equation 5
7: $\quad\quad r_{i,k} = w_{i,k} - \delta_i^-(k)$
8: $\quad\quad r_i^{max} = \max(r_i^{max}, r_{i,k})$
9: $\quad$ **until** $\eta_i^+(x_i) = k$
10: **return** r_i^{max}

The algorithm 3 and 2 are nearly identical. The only difference regards the loop termination in line 9: the loop in algorithm 3 iterates over all jobs of T_i activated within x_i. This is because T_i cannot be activated after the *SCR*, since it is completed task.

E. Worst-Case Response Time of Added Tasks

Let us assume that T_i is an added task. Algorithm 4 calculates the worst-case response time of T_i across a scenario change.

Algorithmus 4 worst-case response time of an added task

```
1: r_i^max = 0
2: for all x_i ∈ X_i do
3:     k = 0
4:     repeat
5:         k++
6:         calculate w_{i,k} according to equation 6
7:         r_{i,k} = max(0, w_{i,k} − x_i − φ_a − (k − 1) · p_a)
8:         r_i^max = max(r_i^max, r_{i,k})
9:     until ⌈(w_{i,k} − x_i − φ_a) / p_a⌉_0 ≤ k
10: return r_i^max
```

Similar to the algorithms 3 and 2, algorithm 4 performs a response time calculation for each value of x_i from $\mathcal{X}_i$ (line 2). The computed value of $w_{i,k}$ in each iteration (line 6) is performed by solving equation 6. This equation takes into account that the analyzed task T_i is an added task, thus it can not be activated before a time $\phi_i + x_i$ after the start of the transition busy window. The equation starts with a value of $w^0_{i,k}$ equal to x_i and is solved iteratively. Note that for a large value of ϕ_i, jobs of T_i may fall outside the transition busy window. In this case, as can be seen in the first clause in equation 6, the analyzed task T_i does not contribute to the calculation of $w_{i,k}$.

$$
\begin{aligned}
w^{n+1}_{i,k} \quad = \quad & \min(k, \left\lceil \frac{w^n_{i,k} - x_i - \phi_i}{p_i} \right\rceil_0) \cdot c^{max}_i + \\
& \sum_{\forall T_u \in hp_u(T_i)} \eta^+_u(w^n_{i,k}) \cdot c^{max}_u + \\
& \sum_{\forall T_c \in hp_c(T_i)} \eta^+_c(x_i) \cdot c^{max}_c + \\
& \sum_{\forall T_a \in hp_a(T_i)} \left\lceil \frac{w^n_{i,k} - x_i - \phi_a}{p_a} \right\rceil_0 \cdot c^{max}_a
\end{aligned} \qquad (6)
$$

Line 7 of the algorithm computes the response time of the job $T_{i,k}$ within $w_{i,k}$. In case no jobs of T_i are activated within the transition busy window, the computed response time is equal to 0. Otherwise, the response time of the considered job $T_{i,k}$ is computed by subtracting from $w_{i,k}$ the distance between the transition busy window start and the activation instant of $T_{i,k}$. If additional jobs of T_i could be activated within $w_{i,k}$ (line 9), the calculation is repeated for $k = k+1$ (line 5), otherwise it terminates. The maximum over all $r_{i,k}$ calculated for all values of x_i represents the worst-case response time of T_i across the scenario change (line 8).

VI. Analysis For Multi-Scenario Distributed Systems

So far, we performed the worst-case response time analysis across a *SC* for multi-scenario uniprocessor systems (Section V). In the following, we extend our approach to analyze response times across a *SC* for multi-scenario distributed systems.

Methods performing response time analysis for distributed systems already exist [2] [13] [9]. However, these methods are only suitable for single scenario systems or systems with mutual exclusive scenario executions. Applying these techniques in their current form to the analysis across a *SC* would require merging both scenarios in a single one, during which both task sets are allowed to execute. This may result in very pessimistic calculated response times.

The method we propose for the response time analysis across a *SC* on distributed systems is an adaptation of the compositional analysis methodology presented in [9]. In the next section, we give an overview about the compositional analysis. In Section VI-B, we show how to adapt the compositional analysis to calculate response times across a *SC* for distributed systems.

A. Compositional Analysis

The compositional analysis methodology alternates local scheduling analysis and event model propagation. First, all external event models at the system inputs are propagated along all system paths until an initial activating event model is available for each task. This approach is safe since, on the one hand scheduling cannot change an event model period, and on the other hand, scheduling can only *increase* an event model jitter [15].

After propagating the external event models, global system analysis is performed. A global analysis step consists of two phases [10]. In the first phase local scheduling analysis (best-case and worst-case scheduling analyses) is performed for each resource and event models at task outputs are calculated. The following equation shows the output event model calculation for a given task T_i:

$$
\begin{aligned}
p^{out}_i &= p^{in}_i \\
j^{out}_i &= j^{in}_i + (r^{max}_i - r^{min}_i)
\end{aligned} \qquad (7)
$$

T_i output event model period obviously equals its activation period. The difference between its worst-case and best-case response times (the response time jitter) is added to its activating event model jitter, yielding the output event model jitter.

In the second phase of the global analysis step, all output event models are propagated to the inputs of the next components. It is then checked if the first phase has to be repeated because some activating event models are no longer up-to-date, meaning that a newly propagated output event model is different from the output event models that was propagated in the previous global analysis step. Analysis completes if either all event models are up-to-date after the propagation phase, or if an abort condition, e. g. the violation of a timing constraint has been reached.

B. Compositional Analysis For Multi-Scenario Systems

The main problem when analyzing response times across a *SC* in a multi-scenario distributed system, is that the worst-case response time of a given task under analysis is not necessarily obtained when the task is activated within the transition busy window. This situation is explained using the following example.

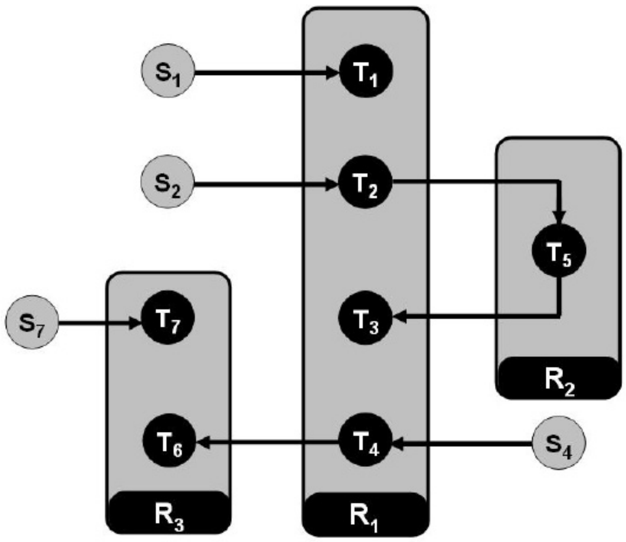

Fig. 4 - DISTRIBUTED SYSTEM EXAMPLE

Figure 4 shows a system example consisting of 7 tasks (T_1 to T_7) mapped on three resources (R_1 to R_3). We assume static priority preemptive scheduling on the resources R_1 and R_3 with priorities assigned as follows: $T_1 > T_2 > T_3 > T_4$ and $T_6 > T_7$. S_1, S_2, S_4 and S_7 represent the event sources at the system inputs. Task T_1 is assumed to be a completed task, T_2, T_3 and T_5 are added tasks, while T_4, T_6 and T_7 are unchanged tasks. The focus will be on the worst-case response time calculation of T_4 across a *SC*. When a *SC* occurs in the system, executions of the completed task T_1 may delay the executions of several jobs of T_2 activated after the *SCR*. This may lead to a burst of events at the output of T_2. This burst of events is then propagated through the task T_5 on R_2 to the task T_3. However, until this burst of events - which is a consequence of the *SC* - arrives at T_3 input, the transition busy window may already be finished on R_1. Therefore, the effect of the transient overload on the resource R_1 due to the *SC* may not be limited to the transition busy window, but may be recurrent. Therefore, for the worst-case response time calculation of T_4 across the *SC*, it may not be sufficient to consider only its jobs activated within the transition busy window, but we may also have to consider its jobs activated within the next busy windows.

The previous example illustrates how difficult it is to predict the effect of the transient overload after a *SCR* on a given resource. As a consequence of this unpredictability it turns to be very difficult to describe the exact timing behavior of events at tasks outputs. Depending on the recurrent transient overload after a *SCR*, the jitter interval size at outputs of unchanged and added tasks may namely vary over time. For instance, jobs of T_4 may be delayed by executions of the higher priority tasks T_1 and T_2 during the transition busy window. This may lead to a burst of events at T_4 output, i.e. to a large calculated jitter interval at T_4 output. After the end of the transition busy window, and before the transient overload effect is propagated to the input of T_3, jobs of T_4 may only be delayed by executions of T_2, resulting in a smaller calculated jitter interval at T_4 output. When the transient overload effect arrives at T_3 input, T_4 jobs may experience longer response times, resulting again in a large calculated jitter interval at T_4 output.

Since output jitter turns into input jitter of connected components, a variable jitter interval size at T_4 output results in a variable jitter interval size at T_6 input. This may strongly increase the response time analysis complexity on the resource R_3 across the *SC*.

To overcome this problem, we need to calculate a jitter interval for each task, that covers all possible jitter interval sizes at the task output resulting from the the recurrent transient overload after a *SC* from an old to a new scenario. This approach is safe since a smaller jitter interval is contained in a larger jitter interval. The calculation is performed by extending the compositional methodology presented in Section VI-A in the following way.

As usual, all external event models in the system inputs are propagated along all system paths until an initial activating event model is available for each task. Then, global system analysis is performed. In the first phase, two local scheduling analysis types are performed for each resource: first, we perform the classical scheduling analysis that assumes mutual exclusive scenario executions for the old and the new scenario. Then, we perform the scheduling analysis that calculates the task response times across the *SC* during the transition busy window as shown in Section V (note that for the best-case scheduling analysis, the task minimum core executions times could be used as best-case response times). From the obtained response times, we calculate for each task a response time interval in which all its observable response times may fall into (i.e. as well its response times during the transition busy windows as its response times assuming mutual exclusive scenario executions). Using this calculated response time interval, we can now calculate for each task, an output period and an output jitter interval, that covers all possible jitter interval sizes at its output. The calculation is performed according to equation 7.

The second step of the global analysis is identical to the one in Section VI-A, i.e. the output event models are propagated to the inputs of the next components and the global analysis is repeated until event models convergence, or if an abort condition has been reached.

VII. Conclusion

The set of executed tasks in modern hard real time systems may change over time. During this change - called scenario change - executions from different task sets may interfere with each other leading to a transient overload situation in the system. Thus, it is necessary to verify that no timing requirements are missed during a scenario change.

In the first part of the paper, we presented a scheduling analysis technique allowing to calculate task worst-case response times across a scenario change in single resource systems under fixed priority preemptive scheduling. While, previous scheduling analysis techniques are limited to purely periodic task activation patterns, we allow more complex activating event models for the tasks. This extension is an essential step towards a scenario aware analysis for distributed systems.

In the second part of the paper, we identified the effects that a scenario change could have in distributed systems on task activation patterns and response times. Then, we proposed a new approach methodology adapted for the response time analysis across a scenario change for multi-scenario distributed systems.

References

[1] R. L. Cruz. A calculus for network delay. *IEEE Transactions on Information Theory*, 37(1):114–141, January 1991.

[2] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123, Oulu, Finland, 1993.

[3] J. J. Gutierrez, J. C. Palencia, and M. G. Harbour. On the schedulability analysis for distributed hard real-time systems. In *Proceedings 9th Euromicro Workshop on Real-Time Systems*, pages 136–143, Toledo, Spain, June 1997.

[4] Marek Jersak. *Compositional Performance Analysis for Complex Embedded Applications*. PhD thesis, Technical University of Braunschweig, 2004.

[5] H. Kopetz and G. Gruensteidl. TTP - a time-triggered protocol for fault-tolerant computing. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, pages 524–532, 1993.

[6] J. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of the Real-Time Systems Symposium*, pages 201–209, 1990.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[8] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197, 2004.

[9] K. Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, Technical University of Braunschweig, 2004.

[10] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor SoC. In *Proceedings 24th International Real-Time Systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[11] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. Technical Report UM-CS-1989-060, 31, 1989.

[12] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design Space Exploration of Network Processor Architectures. In Mark Franklin, Patrick Crowley, Haldun Hadimioglu, and Peter Onufryk, editors, *Network Processor Design Issues and Practices, Volume 1*, chapter 4, pages 55–90. Morgan Kaufmann, October 2002.

[13] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, Geneva, Switzerland, 2000.

[14] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Real-Time Systems. *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[15] 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.

[16] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1992.

[17] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich. SPI – A system model for heterogeneously specified embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4), August 2002.

VIII. Appendix

A. Arrival Function

Table I shows the mathematical representation of the arrival functions defined in Section III-B, using the event model parameters. Notice that, the sporadic class generalizes the class of periodic models, in the sense that only the η^- functions are set to zero. That means, every sporadic event model is in the worst-case a periodic event model.

TABLE I - The arrival functions η^+ and η^-

	$\eta^+(\Delta t)$	$\eta^-(\Delta t)$	
model	periodic & sporadic	periodic	sporadic
simple	$\left\lceil \frac{\Delta t}{p} \right\rceil$	$\left\lfloor \frac{\Delta t}{p} \right\rfloor$	0
w/ jitter	$\left\lceil \frac{\Delta t + j}{p} \right\rceil$	$\max\left(0, \left\lfloor \frac{\Delta t - j}{p} \right\rfloor\right)$	0
w/ burst	$\min\left(\left\lceil \frac{\Delta t + j}{p} \right\rceil, \left\lceil \frac{\Delta t}{d^{min}} \right\rceil\right)$	$\max\left(0, \left\lfloor \frac{\Delta t - j}{p} \right\rfloor\right)$	0

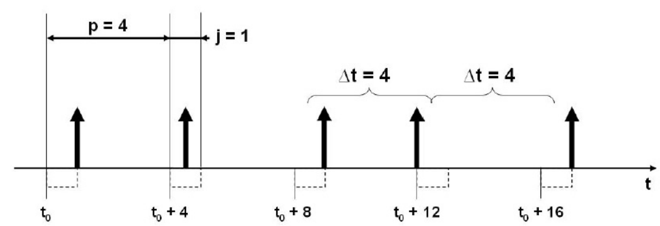

Fig. 5 - Example of a periodic with jitter event model $(p = 4, j = 1)$

For a better understanding of arrival functions, consider an example of a periodic with jitter event model where $(p, j) = (4, 1)$. This event model is visualized in figure 5. Each dashed box indicates a jitter interval of length $j = 1$. The jitter intervals repeat with the event model period $p = 4$. The figure additionally shows a sequence of events which satisfies the event model, since exactly one event falls within each jitter interval box, and no events occur outside the boxes.

Figure 6 shows the graphical representation of the arrival functions corresponding to the periodic with jitter event model

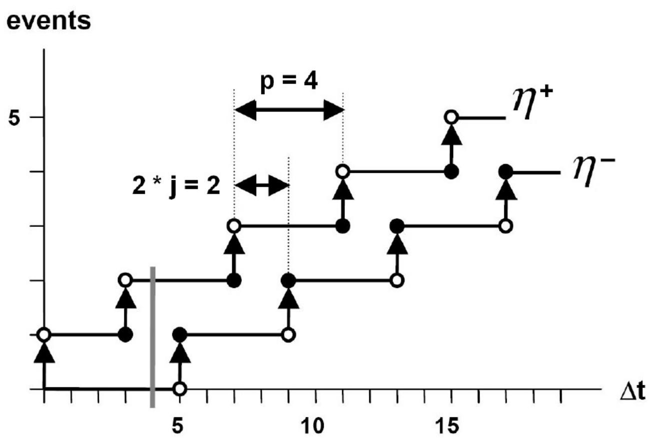

Fig. 6 - Upper and lower arrival functions for the event model $(p = 4, j = 1)$

in figure 5. The event arrival functions are piecewise constant step functions with unit-height steps, each step corresponding to the occurrence of one event. Note that at the points where the functions step, the smaller value is valid for the upper event arrival function, while the larger value is valid for the lower event arrival function (indicated by dark dots). For any time interval of length Δt, the actual number of events is bound by the upper and lower event arrival functions. Event arrival functions resemble arrival curves [1] which have been successfully used by Thiele et al. for compositional performance analysis of network processors [12].

To get a better feeling for event arrival functions, imagine a sliding window of length Δt that is moved over the (infinite) length of an event stream. Consider $\Delta t = 4$ (gray vertical bar in figure 6). The upper event arrival function indicates that at most 2 events can be observed during any time interval of length $\Delta t = 4$. This corresponds e.g. to a window position between $t_0 + 8.5$ and $t_0 + 12.5$ in figure 5. The lower event arrival function indicates that no events have to be observed during $\Delta t = 4$. This corresponds e. g. to a window position between $t_0 + 12.5$ and $t_0 + 16.5$ in figure 5.

B. Distance Function

TABLE II - The distance functions δ^- and δ^+

	$\delta^-(n)$	$\delta^+(n)$	
model	periodic & sporadic	periodic	sporadic
simple	$(n-1)p$	$(n-1)p$	∞
w/ jitter	$\max(0, (n-1)p - j)$	$(n-1)p + j$	∞
w/ burst	$\max((n-1)d^{min}, (n-1)p - j)$	$(n-1)p + j$	∞

Table II shows the mathematical representation of the distance functions defined in Section III-B, using the event model parameters. Note that the distance function is the inverse of the arrival function.

In the the periodic with jitter event model in figure 5, the minimum distance between 2 events is 3 time units (e.g. the distance between the event activated at $t_0 + 9$ and the event activated at $t_0 + 12$), and the maximum distance between 2 events is 5 time units (e.g. the distance between the event activated at $t_0 + 12$ and the event activated at $t_0 + 17$).