Improved Output Jitter Calculation for Compositional Performance Analysis of Distributed Systems

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

Abstract

Compositional performance analysis iteratively alternates local scheduling analysis techniques and output event model propagation between system components to enable performance analysis of heterogeneous distributed systems. In spite of its high scalability and adaptability, the compositional approach may suffer from overestimated results compared with other system performance verification techniques. The main reason is an incomplete consideration of event sequence correlations. In this paper we present a new technique that improves the output jitter calculation by correlating jitter and response times and offers significantly tighter analysis bounds.

1. Introduction

As a consequence of the growing complexity of modern embedded systems, large heterogeneous distributed and multi-core architectures are replacing traditional single-processor architectures thus, increasing the need for performance verification techniques adapted to this new trend. Beyond simulation, there are novel approaches to formal schedulability analysis that scale to large heterogeneous systems and are meanwhile used in industrial practice [12]. From the literature, two different formal approaches addressing this problem can be identified: the holistic approach that extends the classical scheduling theory to distributed systems [14] [2], and the compositional approach that couples local scheduling analysis techniques either using new event models or based on existing event models and analysis techniques [1] [13] [11].

Despite its better ability to take global system effects into account, the holistic performance analysis approach lacks flexibility due to the need to adapt scheduling analysis to each potential system configuration. On the other hand, the compositional performance analysis approach which overcomes these mentioned holistic approach restrictions, in its current form [11] ignores event parameter correlations, such as the correlation between jitter and response time of individual events that is intrinsically considered in holistic analysis. Hence, in complex systems, compositional performance analysis may suffer from pessimism due to an overestimated jitter calculation at component outputs. This overestimation can drastically increase when propagating the event models along the system paths, leading to unnecessarily pessimistic analysis results. To overcome this problem, we propose a new technique allowing a more accurate output jitter calculation by taking into account the above mentioned jitter and response time correlations. We show the applicability of our approach for two well-known scheduling strategies.

In the following section, we will review in detail the existing approaches from the literature that enable performance analysis for heterogeneous distributed systems. In Section 3.1, we introduce our application model. Standard event models are described in Section 3.2. In Section 4, we show the disadvantage of the current technique and present a new approach that improves the output jitter calculation. In Sections 5 and 6 we demonstrate the applicability of our approach for two popular scheduling strategies: Static Priority Preemptive and TDMA scheduling. In Section 7, we apply our technique to analyze the performance of a hypothetical system example. Finally, we interpret the results and draw conclusions.

2. Related Work

Numerous scheduling analysis algorithms for processors and communication 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 (RMA) [7]. Since then, various scheduling analysis algorithms have been developed. Examples include Static Priority Preemptive (SPP) [15] and time-slicing mechanisms like TDMA or Round-Robin [6].

Scheduling analysis algorithms are however not directly

1-4244-0910-1/07/$20.00

applicable to heterogeneous distributed systems with different scheduling and resource-sharing strategies. Formal approaches adapted to such systems are needed. From the literature two known approaches address this problem: the *holistic* and the *compositional* approaches.

The *holistic* approach [14] [2] systematically extends the classical scheduling theory to distributed systems e.g. Tindell have extended existing techniques to allow performance analysis of special heterogeneous architectures: fixed priority-scheduled CPU connected via a TDMA-scheduled bus [14]. In [9] and [8] Eles et al. consider a larger variety of mixed time/event-triggered distributed systems, and focus on real-world communication protocols. One major advantage of the holistic approach over the compositional approach is that it has a global view of the system allowing it to take global performance effects into account to yield to tighter calculated analysis bounds. However, because of the very large number of dependencies, the complexity of the equations underlying the analysis grows with system size and heterogeneity. In practice, the holistic approach is limited to those system configurations which simplify the equations, such as deterministic TDMA networks. It is difficult to define a general procedure to set-up and solve the holistic equations for arbitrary systems.

The *compositional* approach [1] [13] [11] establishes a different view on scheduling analysis allowing performance analysis for arbitrarily complex architectures, thus overcoming the restrictions of the holistic approach. This is done by coupling different existing analysis techniques on individual components via event streams. In the compositional approach, 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. Gresser [1], Thiele [13] and Richter [11] use different event stream representations to describe the timing of the event streams. Gresser [1] uses a superpositional *event vector system*, which is then propagated using complex event dependency matrices. Thiele et. al. [13] use *numerical* upper and lower bound event *arrival curves* for event streams, and similar *service curves* for execution modeling. Because they introduce new stream models, both Thiele and Gresser have to develop new scheduling analysis algorithms for the local components that utilize these models; the host of existing work in real-time system can not be re-used. Furthermore, the new models are far less intuitive than the ones known from the classical real-time systems research, e. g. the model of rate-monotonic scheduling with its periodic tasks and worst-case execution times.

Richter [11] uses parameterized *standard event models* (Section 3.2) from real-time systems research rather than introducing new complex event stream representations. Periodic events or event streams with jitter and bursts are examples of standard event models. Therefore, rather than developing new local analysis techniques, Richter's approach benefits from the host of work in real-time scheduling analysis. However, since the tasks response time calculation strongly depends on the tasks activating event models, the compositional analysis using standard event models requires an accurate calculation of the output event models at the tasks outputs to avoid pessimistic system performance estimates.

In [4] and [10] a technique improving the output jitter calculation by considering the effects of the best-case response time on the output event streams has been presented. In this paper, we present a new technique allowing a more accurate output jitter calculation by considering the correlation between the individual jitter delays of events in input streams and the resulting task response times.

3. System Model

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

3.1. 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 is characterized by its *core execution time* interval (CET interval), defined as the minimum and maximum times the task 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[5].

One execution of a task is called job. $T_{i,k}$ denotes the k-th job of task T_i. 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 activating event data from its input FIFO and writes data into the input FIFOs of dependent tasks. 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. This assumption is 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 3.2.

A task needs to be mapped onto 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 from the set of active tasks according to some scheduling policy. For each task T_i, *scheduling analysis* calculates worst-case (R_i^w) and also best-case (R_i^b) task response times, i.e. the maximum and minimum times between the event arrival at task input and task completion. Scheduling analysis guarantees that all observable response times will fall into the calculated $[R_i^b, R_i^w]$ 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.

3.2. Standard Event Models

The timing of the events within an event stream is described using *event models*. We use a set of six parameterized event models that slightly extend the known event models from literature to capture the consequences of jitters, and subsequently bursts [11]. 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 some of the three parameters need to 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 between events.

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. Many applications use periodic event models to specify the activation pattern of the tasks. Examples can be found 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. A more detailed description of the output jitter calculation can be found in the following section. 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 is used to additionally specify the timing of the activating events in case of bursts, and to reduce the transient load peaks.

In this paper we use J_i^{in} and J_i^{out} to denote the jitter parameter of the input and output event models of task T_i. Additionally, $J_{i,k}^{in}$ and $J_{i,k}^{out}$ represent the input and output jitter delays corresponding to the k-th job of task T_i.

4. Problem Formulation

Most worst-case response time analysis techniques are based on the creation of the longest *busy window*. In general, the busy window is defined as a time interval in which the resource is busy processing tasks mapped on it. The worst-case response time of a task is obtained by comparing the response times of all its jobs that are activated within the longest busy window.

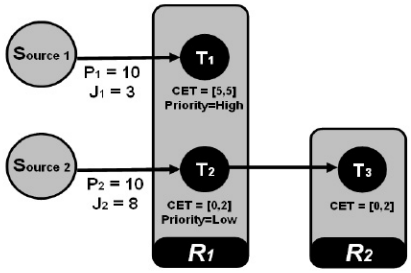

Figure 1. system example

Observe the system in Figure 1 consisting of three tasks T_1, T_2 and T_3 mapped onto two resources R_1 and R_2. We assume SPP scheduling on R_1 and assume that T_1 has the highest priority. The core execution time interval of T_1 is assumed to be $[5,5]$. Both core execution time intervals of T_2 and T_3 are assumed to be $[0,2]$. Task T_1 is activated by events sent by the source task *Source*1, while T_2 is activated by events sent by the source task *Source*2. Let the event model at the input of T_1 be $(P_1^{in} = 10, J_1^{in} = 3)$ and the event model at the input of T_2 be $(P_2^{in} = 10, J_2^{in} = 8)$. In the following, the focus will be on the response time and the output jitter calculation of T_2.

According to [15], the worst-case response time scenario of T_2 is obtained, as shown in Figure 2, by activating T_1 and T_2 simultaneously after a maximum input jitter delay (the jitter interval is represented by a dashed line). The instant t_c at which both tasks are activated is called *critical instant*. All subsequent jobs of both tasks are then activated at or as soon as possible after the critical instant. Note that the busy window starts at the critical instant and finishes when the resource becomes idle i.e. after the job $T_{2,3}$ finishes its execution. The calculated worst-case response time of T_2 is $R_2^w = 12$. It is experienced by the job $T_{2,2}$ whose execution is delayed by the executions of $T_{1,1}$, $T_{2,1}$ and $T_{1,2}$. The calculated best-case response time of T_2 is $R_2^b = 0$.

Now having calculated the response time interval of task T_2 ($[R_2^b, R_2^w] = [0,12]$), we can derive the output event model to its output according to [11]. The output period obviously equals the input period. The output jitter is mathematically calculated using equation 1. It is obtained by composing the input jitter with the jitter resulting from the variation of the task response times. Since the task activation is event-driven, the activation delay at task input is entirely propa-

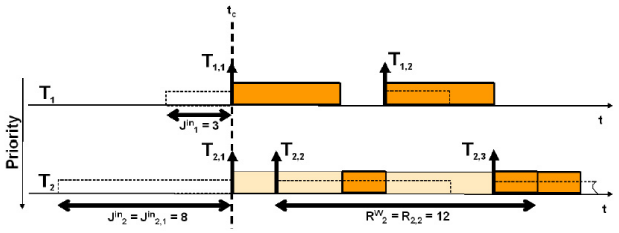

Figure 2. worst-case response time calculation for task T_2

gated at its output, and therefore, the task output jitter is at least equal to the task input jitter. In the above example, the calculated output event model of T_2 is $(P_2^{out} = 10, J_2^{out} = 20)$.

$$J_i^{out} = J_i^{in} + R_i^w - R_i^b \tag{1}$$

The output event model of T_2 turns into the input event model of task T_3. We calculate the worst-case response time of T_3 by applying the approach used above. This calculation is shown in Figure 3: $R_3^w = R_{3,3} = 6$.

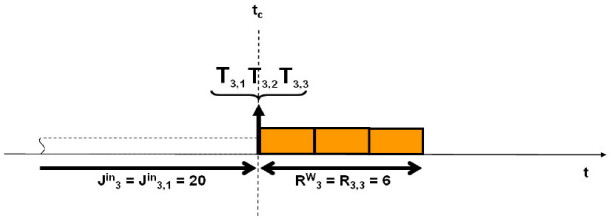

Figure 3. worst-case response time calculation for task T_3

Let us now take a closer look on the output jitter calculation as illustrated in equation 1. It can be observed that the equation associates the maximum input jitter of a task with its worst-case response time. This is only true if the *job* arriving after maximum input jitter delay also experiences the task worst-case response time. However, a closer look on the Gantt charts in Figure 2, shows that $T_{2,1}$, whose activation occurs after maximum input jitter delay $J_{2,1}^{in} = J_2^{in} = 8$, experiences the response time $R_{2,1} = 7$ smaller than the task worst-case response time $R_2^w = 12$. On the other hand, $T_{2,2}$, which experiences the task worst-case response time, is activated after an input jitter delay $J_{2,2}^{in} = 0$ smaller than $J_2^{in} = 8$. This shows that the output jitter calculation in equation 1 might be overestimated, since it ignores the correlation between the input jitter delay and the response time of every job. As the worst-case response time calculation directly depends on the input jitter, the overestimated output jitter calculation is propagated to the next component and may easily lead to an increasingly larger overestimation along the entire application path (in our example the path containing T_2 and T_3).

To overcome this problem, equation 1 has to be modified to consider the correlation between the individual input jitter delays and the resulting response times of the jobs. This is illustrated in equation 2.

$$J_i^{out} = \max_{\forall k}(J_{i,k}^{out}) = \max_{\forall k}(J_{i,k}^{in} + R_{i,k}) - R_i^b \tag{2}$$

Equation 2 requires to find out the execution scenario of the job of T_i with the maximum output jitter delay, i.e., the job of T_i for which the sum of its individual input jitter delay and its response time is maximized. In the following sections, we show for two popular scheduling strategies, that this job belongs to the busy window corresponding to the worst-case response time scenario.

5. Output jitter calculation under static priority preemptive scheduling

Tasks scheduled under SPP are assigned unique priorities. The execution of a lower priority task can be interrupted at any time by the execution of higher priority tasks mapped on the same resource.

5.1. Worst-Case Response Time Scenario

In this section we review the worst-case response time scenario under SPP scheduling as presented by Tindell in [15].

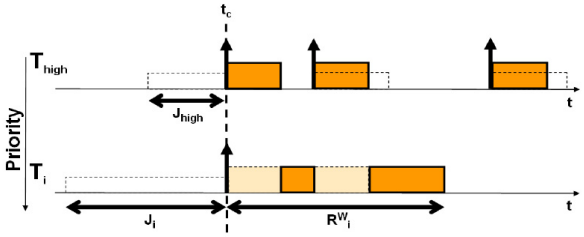

Figure 4. worst-case response time scenario under SPP scheduling

Let T_i be a task scheduled under a SPP scheduler. Let $hp(T_i)$ be the set of higher priority tasks sharing the resource with T_i. The worst-case response time scenario of T_i is shown in Figure 4. It is obtained by simultaneously activating T_i and all tasks belonging to $hp(T_i)$, in our example T_{high}. The instant t_c at which both tasks are activated is called *critical instant*. The first job of each task is activated at the latest possible activation time, which is after a maximum input jitter delay. All subsequent jobs are activated at or as soon as possible after the critical instant. The interval of time starting at the critical instant and during which the

resource is busy processing jobs from T_i or from tasks belonging to $hp(T_i)$ is called a *busy window*. The worst-case response time of T_i is obtained by comparing the response times of all its jobs that execute within the busy window. In the following, we refer to the busy window corresponding to the worst-case response time scenario of T_i with *maximum busy window*.

5.2. Maximum Output Jitter Delay

In this section, we show that the output jitter delay corresponding to the k-th job of T_i executing within the maximum busy window is larger than or equal to the output jitter delay corresponding to the k-th job of T_i executing within any other busy window.

It is obvious that the worst-case response time scenario presented in Section 5.1 maximizes both the number and the response times of the jobs of T_i released within a busy window. From this characteristic, we derive the following lemma.

Lemma 1. *The* k*-th job of T_i released within the maximum busy window experiences a response time which is larger than or equal to the response time experienced by the* k*-th job of T_i released within any other busy window.*

From lemma 1, we can easily derive the following lemma.

Lemma 2. *The output jitter delay corresponding to the first job of T_i released within the maximum busy window is larger than or equal to the output jitter delay corresponding to the first job of T_i executing within any other busy window.*

Proof. The first job of T_i within the maximum busy window is released after a maximum input jitter delay, i.e. $j_{i,1}^{in} = j_i^{in}$. In addition, according to lemma 1, its response time is larger than or equal to the response time experienced by the first execution of T_i within any other busy window. Therefore, its output jitter delay calculated according to equation 2 is maximized. □

Before, we generalize lemma 2 for all jobs of T_i released within the maximum busy window, we introduce the following lemma.

Lemma 3. *Let $T_{i,k}$ be the* k*-th job of T_i released within a given busy window, where $k > 1$. As long as $T_{i,k}$ is released within the busy window, moving its activation within its jitter interval does not affect its corresponding output jitter delay.*

Proof. Let $J_{i,k}^{in}$ be the input jitter delay and $R_{i,k}$ the response time experienced by $T_{i,k}$. If we cause the arrival of the event activating $T_{i,k}$ to occur later within its jitter interval, the input jitter delay $J_{i,k}^{in}$ is increased while decreasing the response time $R_{i,k}$ by the same amount of time. If now we cause the arrival of the event activating $T_{i,k}$ to occur earlier within its jitter interval, the input jitter delay of $J_{i,k}^{in}$ is decreased while increasing the response time $R_{i,k}$ by the same amount of time. Therefore, the calculated output jitter delay corresponding to $T_{i,k}$ according to equation 2 would be always the same when moving the activation of $T_{i,k}$ within its jitter interval. This can be observed by comparing the Gantt charts in Figure 5 and Figure 6. □

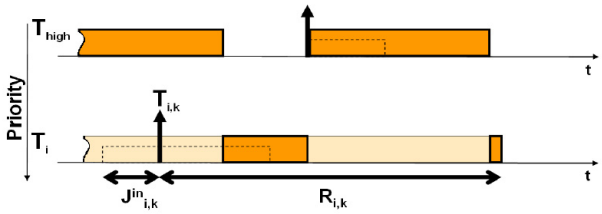

Figure 5. early activation of $T_{i,k}$ within its jitter interval compared with its activation in figure 6

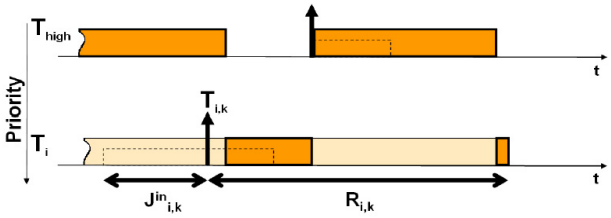

Figure 6. late activation of $T_{i,k}$ within its jitter interval compared with its activation in figure 5

Now, using the previous lemmas we can prove the following lemma.

Lemma 4. *The output jitter delay corresponding to the* k*-th job of T_i released within the maximum busy window is larger than or equal to the output jitter delay corresponding to the* k*-th job of T_i executing within any other busy window*

Proof. Lemma 4 is already proven for $k = 1$ (lemma 2). Let now $T_{i,k}$ be the k-th job of T_i within a given busy window, where $k > 1$. Let $J_{i,k}^{in}$ be the input jitter delay, $R_{i,k}$ the response time and $J_{i,k}^{out}$ the output jitter delay corresponding to $T_{i,k}$. According to lemma 1, $R_{i,k}$ is smaller than or equal to the response time of the k-th job of T_i released within the maximum busy window. If in addition $J_{i,k}^{in}$ is smaller than or equal to the input jitter delay of the k-th job of T_i within the maximum busy window, it is obvious that lemma 4 is proven for $T_{i,k}$. Let us now assume that $J_{i,k}^{in}$ is larger than the input jitter delay experienced by the k-th job of T_i within the maximum busy window. According to lemma 3, if we move the

activation of $T_{i,k}$ to occur earlier within its jitter interval, $J_{i,k}^{out}$ is not affected. In particular, if we activate $T_{i,k}$ after an input jitter delay equal to the input jitter delay experienced by the k-th job of T_i within the maximum busy window, the value of $J_{i,k}^{out}$ does not change. Therefore, since $R_{i,k}$ is smaller than or equal to the response time of the k-th job of T_i within the maximum busy window, $T_{i,k}$ satisfies lemma 4. □

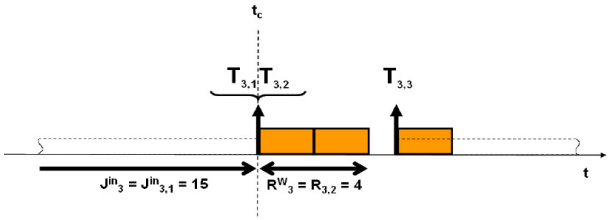

Figure 7. improved worst-case response time calculation for task T_3

From lemma 4 we conclude that under SPP, the maximum output jitter delay corresponding to a job of T_i is obtained by comparing the individual output jitter delays of all jobs of T_i that are released within the maximum busy window. let us apply this for the system in Figure 1. When applying equation 2, we calculate the following output jitter delays for the three jobs of T_2 activated within the busy window: $J_{2,1}^{out} = 15$, $J_{2,2}^{out} = 12$ and $J_{2,3}^{out} = 4$. Therefore, the output jitter of T_2 is provided by the job $T_{2,1}$: $J_2^{out} = J_{2,1}^{out} = 15$. After propagating the new calculated output event model to the input of task T_3, we calculate, as shown Figure 7, a tighter worst-case response time for T_3: $R_3^w = R_{3,2} = 4$.

6. Output jitter calculation under time division multiple access scheduling

Each task scheduled under TDMA is assigned a time slot. The sequence of consecutive time slots of all tasks mapped on a TDMA resource is called *round* or *turn*. Tasks can only execute within their respective time slots. If a task does not finish executing before the end of its time slot, it has to wait for its time slot in the next round.

6.1. Worst-Case Response Time Scenario

In this section we review the worst-case response time scenario under TDMA scheduling as presented in [6].

Let T_i be a task scheduled under TDMA scheduling. The worst-case response time scenario of T_i is shown in Figure 8. It is obtained by activating T_i at the end of its time slot, after having experienced a maximum input jitter delay. The instant at which T_i is activated is called critical instant. All subsequent jobs of T_i are then activated as soon as possible after the critical instant. Under a TDMA scheduler, we define the busy window as the interval of time starting at the

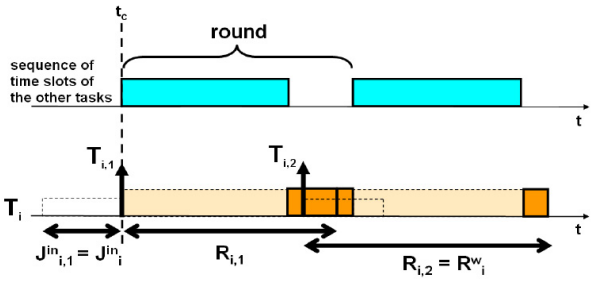

Figure 8. worst-case response time scenario under TDMA scheduling

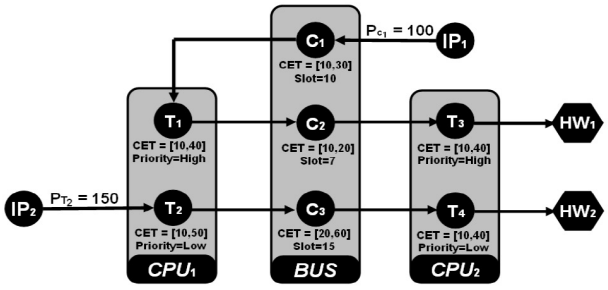

Figure 9. heterogeneous distributed system example

critical instant until the first instant at which the resource becomes idle within a time slot of T_i. The worst-case response time of T_i is obtained by comparing the response times of all its jobs that are activated within the busy window. In the example in Figure 8, the worst-case response time is provided by the job $T_{i,2}$.

6.2. Maximum Output Jitter Delay

From the point of view of a task scheduled under TDMA, the sequence of time slots of the other tasks in a round could be assimilated with a periodic execution of a higher priority task under SPP scheduling. Therefore, the argumentation presented in Section 5.1 could be applied to show that the busy window corresponding to the worst-case response time scenario under TDMA scheduling contains the job that maximizes the output jitter delay according to equation 2.

7. Distributed System Example

In the following, we apply our output jitter calculation technique presented in the preceding sections to analyze the performance of a hypothetical heterogeneous distributed system.

Figure 9 shows an example of a heterogeneous architecture, containing two processing elements (CPU_1,CPU_2) that communicate via a shared bus (BUS). A periodic data

stream coming from the IP_1 component travels through the bus on channel C_1 and then activates task T_1 on CPU_1. When task T_1 finishes execution, it transmits data via channel C_2 to task T_3 running on CPU_2. T_3, in turn, sends data to the co-processor HW_1. In parallel, another periodic stream coming from the IP_2 component activates the execution of task T_2 mapped on CPU_1. At its completion, task T_2 sends a data stream through channel C_3 to task T_4 on CPU_2. After finishing its execution T_4 transmits data to the connected hardware component HW_2.

As we can see, both data streams (the one traveling from IP_1 to HW_1 and the other traveling from IP_2 to HW_2) are crossing different common resources ($CPU1, CPU_2$ and BUS).

The IP_1 periodically sends data packets with a period of $100\mu s$. The IP_2 component also sends data blocks with a period of $150\mu s$. The processing resources CPU_1 and CPU_2 are running a SPP scheduler. The communication channels C_1, C_2, and C_3 share the BUS using a TDMA arbitration scheme. Table 1 provides the timing and scheduling parameters of the tasks running on CPU_1 and CPU_2. Table 2 provides the timing and scheduling parameters of the channels running on BUS. The time values are expressed in μs.

Table 1. tasks parameters

CPU_1 Static Priority Preemptive			CPU_2 Static Priority Preemptive		
Task	CET	Priority	Task	CET	Priority
T_1	[10;40]	high	T_3	[10;40]	high
T_2	[10;50]	low	T_4	[10;40]	low

Table 2. channels parameters

BUS TDMA		
Channel	Access Time	Time Slot
C_1	[10;30]	10
C_2	[10;20]	7
C_3	[20;60]	15

In the following, we apply the known local scheduling analysis, then determine the output event models, and following the compositional approach, propagate these to the inputs of the connected components. Notable is that the analysis on CPU_1 requires the input event model of task T_1. However, this cannot be calculated before we analyze the communication on the bus. For the communication analysis we need the input event models of channels C_2 and C_3 that are generated only after we analyze CPU_1. Therefore, there is cyclic dependency between CPU_1 and the bus. This problem is solved by initially propagating all external event models along all system paths until an input event model is available for each task. Then, local scheduling analysis and event model propagation are performed alternately 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 [3].

Table 3 shows the response times and the jitter values of the input and output event models calculated using equation 1.

Table 3. previous analysis results considering the jitter calculation as illustrated in equation 1

Task	Response Time	Input Jitter	Output Jitter
T_1	[10;66]	86	142
T_2	[10;170]	0	160
C_1	[10;96]	0	86
C_2	[35;227]	142	334
C_3	[37;246]	160	369
T_3	[10;65]	334	389
T_4	[10;409]	369	768

We see that the jitter drastically increases along a dependency path. When we consider the path $IP_2 - T_2 - C_3 - T_4 - HW_2$, we observe that the periodic data stream coming from the IP_2 component turned into a heavy burst event stream at HW_2 input. A periodic with jitter event model with the period equal to 150 and the jitter equal to 768 may have at the input of HW_2 in the worst-case scenario a burst activation with the burst size equal to 6. Such a burst obviously generates a high transient load.

Let us apply the analysis on the system again, but this time using the output jitter calculation as expressed in the equation 2. Table 4 shows the result values obtained from the analysis.

Table 4. new analysis results considering the jitter calculation as illustrated in equation 2

Task	Response Time	Input Jitter	Output Jitter
T_1	[10;66]	86	116
T_2	[10;170]	0	160
C_1	[10;96]	0	86
C_2	[35;201]	116	176
C_3	[37;246]	160	251
T_3	[10;50]	176	206
T_4	[10;246]	251	441

Notable is that the jitters along the paths do not increase that fast when applying the improved output jitter calculation technique. Furthermore, the calculated worst-case response times of the tasks are noticeably shorter. This is because a better jitter estimation at the output of a task strongly reduces the uncertainty at the input of subsequent tasks. Table 5 shows the calculated output jitter and response time

reductions when comparing the results obtained using the improved output jitter calculation technique with the results obtained using equation 1. As it can be observed, there is no calculation improvement at the outputs of T_2 and C_1 since these tasks are activated strictly periodically (jobs of these tasks are activated without any input jitter delay). It is also notable that the calculation improvement increases continuously along the system paths.

Table 5. output jitter and response time reduction

Task	Output jitter Reduction (%)	Resp. Time Reduction (%)
T_1	18.3	0
T_2	0	0
C_1	0	0
C_2	47.3	11.5
C_3	31.9	0
T_3	47.0	23.0
T_4	42.5	39.8

8. Conclusion

Accurate output jitter calculation is essential to apply efficiently the compositional approach to the performance analysis of heterogeneous distributed systems. Output jitter turns namely, into input jitter of connected components and strongly affects their response time calculation. In this paper, we identified the limitations of the existing output jitter calculation technique and presented a new approach improving the jitter bound at task outputs. We also showed that our technique simply requires the calculation of the output jitter delays of the jobs that are considered in the activation scenarios leading to the task worst-case response time. The experiments performed on a distributed system example showed that the new technique significantly improves the overall system performance bounds.

References

[1] 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.

[2] 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.

[3] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the symta/s approach. *IEE Proceedings Computers and Digital Techniques*, 152(2):148–166, March 2005.

[4] J. J. Gutirrez J. C. Palencia and M. G. Harbour. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In *Proc. of the 10th Euromicro Workshop on Real-Time Systems*, 1998.

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

[6] 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.

[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] P. Pop, P. Eles, and Z. Peng. Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. In *Proc. Design, Automation and Test in Europe (DATE 2003)*, Munich, Germany, 2003.

[9] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proc. Int. Symposium on Hardware/software codesign CODES'02*, Estes Park, USA, 2002.

[10] R. Racu, K. Richter, and R. Ernst. Calculating task output event models to reduce distributed system cost. In *Proceedings of GI/ITG/GMM-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 1–10, Kaiserslautern, Germany, February 2004.

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

[12] Symtavision. http://www.symtavision.com.

[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.