Calculating Task Output Event Models to Reduce Distributed System Cost

Razvan Racu, Kai Richter, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig, Germany
{racu, richter, ernst}@ida.ing.tu-bs.de

Abstract

Input event models such as periodic, jitter, and sporadic are known as powerful abstractions for task workload arrival in scheduling analysis. In distributed systems, such task input models result from output event models of other tasks in the system. Interestingly, output event models have received only little attention in literature. In this paper, we introduce a novel approach to determine task output event models for the most practically important task configurations. We show that the output event model accuracy is key to efficient system dimensioning with a direct impact on system cost, and we propose several optimizations. The experiments show that the new approach reveals previously unknown, and hence, unexploited system performance reserves.

1. Introduction

Performance and timing analysis, specifically with respect to scheduling influences, is a known problem in distributed system design. The traditional real-time scheduling analysis uses very efficient environment models, or event models, to capture the arrival of workload. The RMS [5] is a popular example, analyzing periodically arriving tasks. However, the known approaches are limited to a single CPU or bus but fail to analyze more complex distributed systems. Multiprocessor extensions, called holistic approaches, can cope with slightly more complex architectures, but their applicability and scalability is still limited.

In a recent approach, called Symta/S, event streams are used to couple several existing local techniques into a system-level analysis model. An event stream connects two locally analyzable components, such that the output event model of one component becomes the input event model for a connected component. But while input event models are well known and have been used for work load analysis, output event models have received only little attention because they are seemingly less interesting.

Output event models describe the generation of workload, such as production entering a bus. The producer determines the arrival of packets at the bus. First, we have to consider how often that producing task is executed. This information is given by the tasks activation or input event model. Secondly, we need to consider that a task does not necessarily always produce a packet on each execution. In other words, the task output can be conditional. For instance, a periodic task execution can lead to non-periodic – or sporadic – packet generation. In order to correctly determine the output event model, such conditional task behavior needs to be considered.

Finally, the execution time of a task represents a delay between input and output, and operating systems usually disturb the tasks execution further, e. g. a periodically scheduled task does not neces-

sarily execute periodically, but can experience a varying number of preemptions. This obviously adds jitter characteristic to the execution behavior, and subsequently, to the output generation.

The mentioned influences, namely input model dependency, conditional output generation and jitter injection, can result in complex output event models. These models then become input models for connected tasks and determine communication load. Hence, the output event model determination needs to consider such complex models at component or communication inputs.

In this paper, we introduce an approach to derive best-case and worst-case output event models for several task configurations. A task configuration in this sense in given by a) the tasks activation model or input event model, which can be *periodic* or *sporadic*, both with or without jitter, and b) the tasks functional behavior which can be conditional or not. We assume jitter injection due to scheduling for all configurations.

In the next section, we analyze periodic tasks with unconditional behavior. This allows us to start with slightly simplified assumptions. More precisely, we focus on the scheduling influence, which has not yet been considered for output event model calculation. Interestingly, the best-case scheduling can lead to worst-case output generation. Distributed systems are known to exhibit such *scheduling anomalies* [2]. But the scheduling influence on output models has hardly been investigated. So far, only very conservative approaches exist. New and much more accurate output model calculation approaches are presented in Section 3 and 4. They allow to reveal and exploit previously unknown performance reserves. In Section 5, we extend the basic approach to the other configurations, namely conditional tasks, more complex input event models (sporadic with or without jitter) as well as combinations of both. Finally, we draw the conclusions and give an outlook on future work.

2. Output Models of Periodic, Unconditional Tasks

In this section we determine the output event models for tasks with very simple configurations. Consider a task with a periodic activation and a one-to-one correspondence between input (task activation) and output (task completion) and a fixed execution time. Each activation experiences a fixed delay from input to output, and thus, the output event model is equal to the input event model. Due to the input-output delay, the stream offsets are different but the event model does not consider them. Only the relative timing is important when defining event models and in this case it remains unchanged from input to output. But, usually the software processes have a non-constant execution time. Moreover, the tasks are sharing common resources so that the scheduling disturbs the task execution further. So, we obtain a non-constant delay between task activation and task completion. The output event model is not purely periodic anymore. However, the delays are bounded and the upper bound is known from the scheduling analysis theory as *worst-case response time*. The *response time* determines the time span between activation (input event) and completion (output event) of a task. We illustrate this using the following example.

Consider a set of three periodic tasks P_1, P_2 and P_3 running on the same processing resource. The activation periods are $T_1 = 100\mu s$, $T_2 = 240\mu s$ and $T_3 = 500\mu s$. The tasks have the core execution times $C_1 = 40\mu s$, $C_2 = 70\mu s$ and $C_3 = 120\mu s$, respectively. Task P_1 has the highest priority and task P_3 the lowest. Figure 1 shows the Gantt diagrams corresponding to the worst-case activation scenario [5].

The first activation of P_2 is preempted two times by task P_1 and finishes $\Delta t1 = C_2 + 2 \cdot C_1 = 150\mu s$ after the activation. The second activation of P_2 is preempted only once, $\Delta t2 = C_2 + C_1 = 110\mu$. Moreover, the first activation of P_2 occurs at its *critical instant*, and thus, according to the theory from [5], $\Delta t1$ represents the maximum or worst-case response time of P_2. Equation 1 shows the RMS worst-case response time calculation(R_i) of P_i where C_i and T_i represent the core execution time and the activation period of P_i and $HP(i)$ is the set of all higher priority tasks.

$$R_i^{(n+1)} = C_i + \sum_{\forall j \in HP(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil \cdot C_j \leq T_i \tag{1}$$

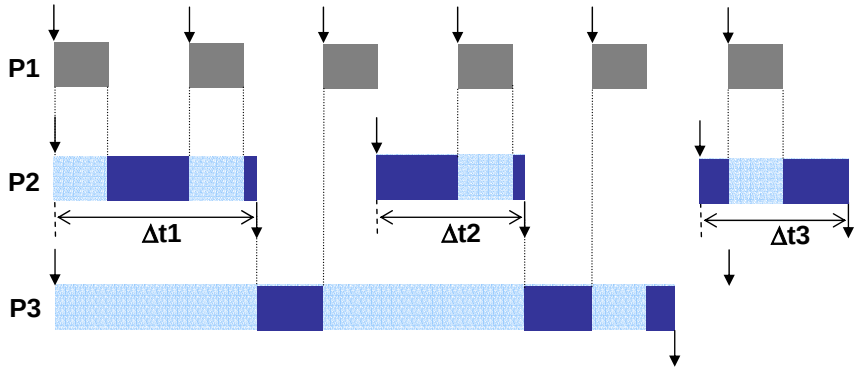

Figure 1. Non-constant delays between task activation and task completion

In a similar way a lower limit can be defined, i. e. the minimum task response time can be determined. Thus, even though the response time is not constant, it is always bounded by an interval. This means the key property of the input stream is preserved in the output stream, i. e. a generally periodic behavior. However, due to a non-constant response time there is some uncertainty about the actual timing of each event. Such event stream behavior is known as *jitter*. A jitter allows each event to slightly deviate from a *reference period*. The jitter bounds the maximum deviation in time. Figure 2 shows the jitter characteristic of the output event model.

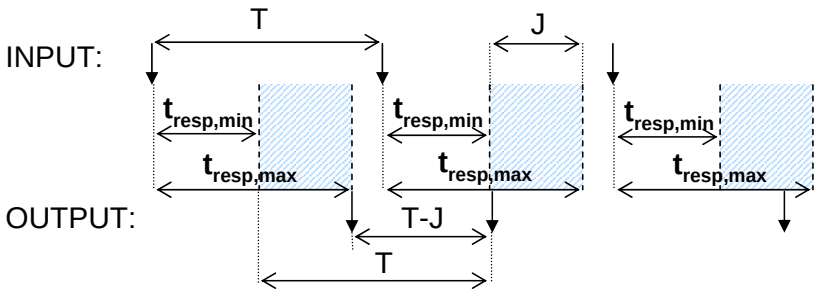

Figure 2. The presence of a jitter at the output

The output jitter is the difference between the highest and the lowest possible execution delay. This is the basic idea to determine the output jitter for the simple task configuration. We have shown above how upper response time bounds can be calculated. Besides the relatively simple RMS [5], there is a host of other approaches capturing more complex task activations. Periodic activation with jitter is considered in [1]. Tasks with arbitrary deadlines are the subject of [4]. Interestingly, the lower response time bound has received only little attention. This is mainly because scheduling analysis traditionally considers only load and response time analysis in the worst-case situations, but no output event models.

In Figure 2, we observe that the worst-case output occurs when two events are very close in time. The first task activation finishes after the maximum response time (late event). The second task release finishes after the minimum response time (early event). This is exactly the worst-case output scenario. The distance between these events is (T-J) while the average period is T. This requires not only the upper (for late event) but also the lower (for early event) response time bound to be known. In other words, the worst-case output depends on the best-case response time.

The dependency between best-case and worst-case in distributed system scheduling is known as *scheduling anomaly* [2]. In the next section, we analyze the best-case response times in detail and propose a set of optimizations.

3. Output Event Model Accuracy

Audsley [1] provided first ideas for distributed scheduling analysis and conservatively bounded the minimum response time to be *zero*. Tindell [9] and Eles [6] exploit these properties in their so called holistic scheduling analysis approaches. However, a zero response time is, as we will see soon, a very conservative bound, resulting in very pessimistic performance estimates.

A zero response time will obviously never occur. Even if we totally neglect the influence of scheduling, i. e. we assume no preemption, the minimum response time is at least the task core execution time. And the core execution time is already known since it is essentially required for scheduling analysis. While the maximum core execution time is required for worst-case response time analysis, the minimum core execution time is used to compute the lower response time bound. Both values can be determined using approaches such as [10].

We carried out a set of experiments to demonstrate that using the minimum core execution time as the minimum response time instead of zero results in a significant accuracy gain in the output event models.

Figure 3 shows an example of a heterogeneous architecture, containing two processing elements (CPU1, CPU2) that communicate via a shared bus or network (NoC). A periodic data stream coming from the IP1 component travels through the network on channel C_1 and then activates task P_1 on CPU1. When task P_1 finishes execution, it transmits data via channel C_2 to task P_3 running on CPU2. P_3, in turn, sends data to the coprocessor HW1.

In parallel, another periodic stream coming from the IP2 component activates the execution of task P_2 mapped on CPU1. At its completion task P_2 sends a data stream through channel C_3 to task P_4 on CPU2. After execution P_4 transmits data to the connected hardware component HW2.

As we can see, both data streams (the one traveling from IP1 to HW1 and the other traveling from IP2 to HW2) are crossing different common resources like CPU1, CPU2, and NoC. As explained above both streams experience complex delays due to resource scheduling.

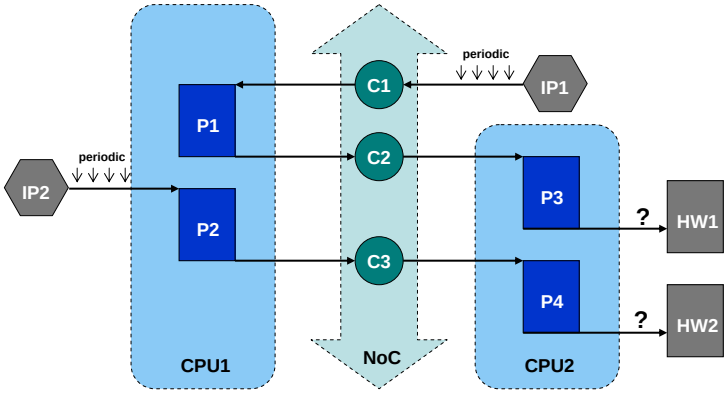

Figure 3. Event streams connecting system components

The IP1 periodically sends data packets with a period of $100\mu s$. The IP2 component also sends data blocks with a period of $150\mu s$. Both processing resources CPU1 and CPU2 are running a static priority preemptive scheduling. The communication channels C_1, C_2, and C_3 share the network resource NoC using a TDMA arbitration scheme. Table 1 provides the timing and scheduling parameters of the tasks and channels running on CPU1, CPU2 and NoC. The time values are expressed in μs.

We apply the known local scheduling analysis, then determine the output models, and –following the recent Symta/S approach– propagate these to the inputs of the connected components. The analysis requires the input event model of task P_1, but this cannot be found 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 CPU1. We observe that the system contains a dependency cycle between CPU1 and the communication network. In order to start the analysis on CPU1 we assume that the communication network does not add any jitter characteristic to the event stream

Table 1. Task/Channels parameters

CPU1 Static Priority Preemptive			CPU2 Static Priority Preemptive			NoC Time Division Multiple Access		
Task	TCore	Priority	Task	TCore	Priority	Channel	Access time	TSlot
P_1	[20;40]	1 (high)	P_3	[10;25]	1 (high)	C_1	[20;30]	10
P_2	[40;50]	2	P_4	[40;40]	2	C_2	[20;20]	7
						C_3	[30;60]	15

coming from IP1 and traversing the bus through the channel C_1. Thus, the task P_1 will have a periodic activation. With purely periodic input, and thus, task activation, we can use the RMS analysis [5] and we obtain the worst case response times of tasks P_1 and P_2. We consider for all tasks/channels that the best case response time is zero. After CPU1 is analyzed, the input event models for C_2 and C_3 can be computed and we can apply the TDMA analysis on the network [3]. From the analysis results we observe that NoC analysis added jitter to the periodic activation of P_1. So, CPU1 must be reanalyzed, but this time, using an analysis that can cope with input jitter [1] [4]. Now, a larger jitter is computed for C_2, so we must also reanalyze the network. Iteratively applying local analysis techniques and propagating the output streams to the inputs of connected components solves the above mentioned dependency cycle [7].

After two iterations, the jitter values converge and the iteration is terminated. So, we can now perform the analysis on the last processing element (CPU2). Table 2 shows the response times and the jitter values of the input and output event models as well as the jitter induced by scheduling. The output jitter is determined by the sum of the input jitter and the tasks internal jitter.

Table 2. Analysis results considering best case equal zero (μs)

Task	Response Time	Input Jitter	Internal Jitter	Output Jitter
P_1	[0;76]	96	76	172
P_2	[0;170]	0	170	170
C_1	[0;96]	0	96	96
C_2	[0;257]	172	257	429
C_3	[0;256]	170	256	426
P_3	[0;125]	429	125	554
P_4	[0;336]	426	336	762

We see that the jitter drastically increases along a dependency path. When we consider the path IP2-P_2-C_3-P_4-HW2, we observe that the periodic data stream coming from IP2 component turned into a heavy burst event stream at HW2 input. A periodic with jitter event model with the period equal to 150 and the jitter equal to 762 may have in the worst-case scenario burst activation with the burst size equal to 6. Figure 4 a) shows the worst-case activation scenario for this event model. Due to the large jitter (762) the first 6 activations can arrive at the same time. Moreover, the next activations arrive as soon as possible relative to the time instant when the burst activations were released. Such a burst obviously generates a high transient load.

We apply the analysis on the system again, but this time using the minimum core execution time as an approximation of the best-case response time. Table 3 contains the result values obtained from the analysis. We see that the new approach reduces the internal jitters, and thus, the jitter of the output event models. We see that the values for the output jitter decreased by $\sim$28% compared to the values obtained considering the best-case response time equal to zero.

Referring to the same path as the one analyzed in the previous section (IP2-P_2-C_3-P_4-HW2) we observe that the event stream entering HW2 radically changed its parameters. It still has a burst behavior but the burst size decreased from 6 to 4. Figure 4 b) shows the activation diagram corresponding to the output event model of P_4.

We see that the jitters do not increase that fast if we use minimum core time as minimum response time. Furthermore, we see that smaller input jitters result in less distortion due to scheduling, and the worst-case response times of other tasks are noticeably shorter. This corresponds to previously

Table 3. Analysis results considering best case equal minimum core execution time (μs)

	Best Case = minimum core execution time				Best Case = 0	Output Jitter
Task	Response Time	Input Jitter	Internal Jitter	Output Jitter	Output Jitter	Reduction (%)
P_1	[20;56]	76	36	112	172	34,9
P_2	[40;170]	0	130	130	170	23,5
C_1	[20;96]	0	76	76	96	20,1
C_2	[20;197]	112	177	289	429	32,6
C_3	[30;236]	130	206	336	426	21,1
P_3	[10;89]	289	79	368	554	33,6
P_4	[40;270]	336	230	566	762	25,7
					Average	27,4

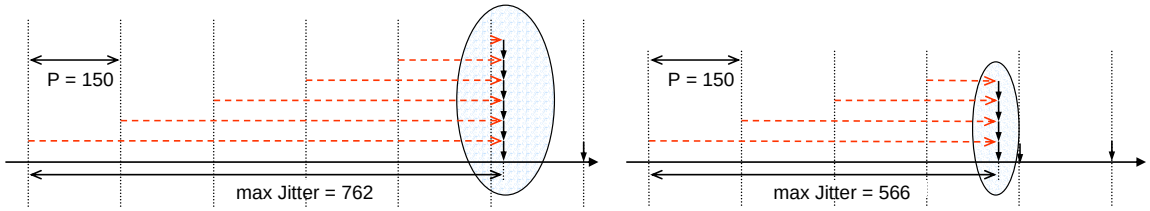

Figure 4. Transient load due to burst activations a) Best Case = 0; b) Best Case = min core exe

unknown performance reserves which can now be exploited.

Suppose that the tasks P_3 and P_4 on CPU2 have the deadlines $D_3 = 130$ and $D_4 = 500$. The deadlines are met in both cases. But we see that the analysis considering the minimum core execution time as the best-case response time reveals previously unknown *performance reserves*. We can now reduce the required performance of CPU2 without violating the deadlines of P_3 and P_4. We carried out additional experiments (without tables) and found that we can reduce the clock speed of CPU2 by 25% without violating the timing constraints of P_3 and P_4. In this case, the worst-case response times of P_3 and P_4 are 129 and 410, respectively, so the deadlines are still met. Reducing the clock speed allows to chose a cheaper CPU. This shows the consequences of analysis deficiencies when output event models are overly conservative.

In the next section we provide a more sophisticated algorithm, aiming at an even more accurate approximation of the minimum response time.

4. Best-Case Scheduling Approximation

In this section we propose a new approach for the improvement of best case scheduling analysis. We developed this approach for static priority preemptive scheduling and for TDMA scheduling.

4.1. Static Priority Preemptive Scheduling Analysis

First, we shortly review the theoretical aspects formulated by Liu and Layland regarding the worst-case response time calculation for static priority preemptive schedulers. They define the critical instant of a task P_i as the time when P_i is preempted by all higher priority tasks in the system. They proved that if P_i is activated at its critical instant then it experiences the longest response time, i.e. the worst-case response time. Moreover, all higher priority tasks must rearrive as soon as possible in order to preempt P_i as often as possible.

An inverse argument can be used for the best case approach: P_i must be activated exactly at the time instant when all higher priority tasks finish their execution, i.e. when the processing resource just becomes idle. Moreover, all higher priority tasks must rearrive as late as possible thus, preempting P_i as rare as possible.

Consider the following example: 3 tasks P_1, P_2 and P_3 are running on the same processing core CPU. The input event models of P_1 and P_2 are periodic with jitter with the parameters $T_1 = 10ms$,

$J_1 = 1ms$ and $T_2 = 11ms$, $J_2 = 1ms$. Task P_3 is periodically activated with the period equal to $100ms$. The core execution times of each process are: $[2ms; 3ms]$ for P_1, $[5ms; 5ms]$ for P_2, and $[11ms; 15ms]$ for P_3. We can see in Figure 5, the Gantt diagrams corresponding to possible best-case scenarios of P_3.

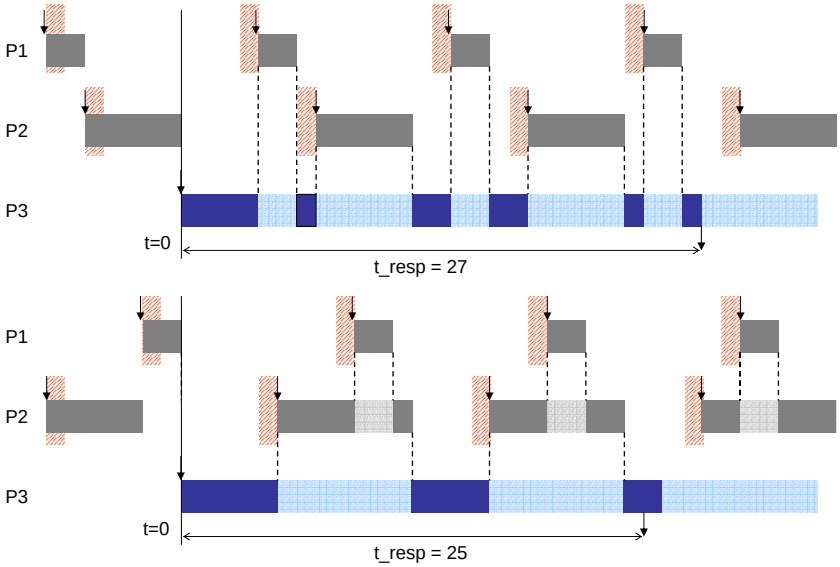

Figure 5. Two candidates for the best-case response time of task P_3

As we can observe, several scenarios (or candidates) have to be investigated in order to get the best case response time of task P_3. They differ in the order in which the higher priority tasks are activated. Obviously, each possible permutation of higher priority tasks must be verified in order to get the minimum response time. Hence, the computational effort grows quickly with the number of higher priority tasks.

We found a very promising approximation that has a linear complexity. Consider the case that all higher priority tasks finish at time $t = 0$. Of course, this scenario is practically impossible since all tasks share the same resource. Figure 6 shows the (assumed) Gantt diagram for this approach. If we compare these with the one shown in Figure 5 we observe that in this approach all higher priority tasks were activated as late as possible (all tasks finish at the same time compared to the real case scenarios when they are executed one after the other). Obviously, they rearrive as late as possible such that there is no realistic scenario where the tasks will rearrive later. Hence, the analyzed task experiences in this scenario the minimum number of preemptions. The formal proof, we have carried out, must be omitted due to space limitations.

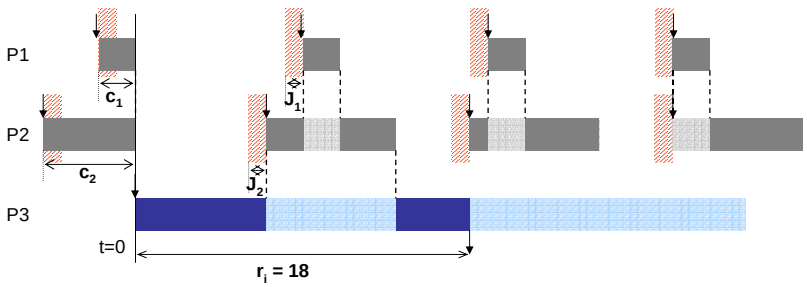

Figure 6. Conservative scenario for the best case response time of task P_3

This approximated best-case response time bound can be iteratively computed using the following

equation:

$$r_i^{(n+1)} = c_i + \sum_{j \in HP(i)} max\left(0, \left\lfloor \frac{r_i^{(n)} + c_j - J_j}{T_j} \right\rfloor\right) \cdot c_j \quad (2)$$

c_i is the minimum core execution time of P_i, T_i and J_i are the activation period and the jitter of P_i and $HP(i)$ is the set of all higher priority tasks.

4.2. TDMA Scheduling Analysis

Similar observations can help to optimize the best case response time analysis for TDMA scheduling. In the previous section we considered the best case response time of P_i, running on a resource with a static priority arbitration scheme, as being equal to its minimum core execution time. This is valid only for the case when the minimum core execution time of P_i does not exceed its time slot. In this situation, it is not preempted. Moreover, in case of TDMA analysis we can obtain exact values for the best-case response times since the tasks execution is independent of the other tasks activation. Each task gets its own resource *slice* that is statically defined.

Thus, in order to construct the best case response time scenario we consider that P_i is activated exactly at the beginning of its time slot and we can easily compute the number of time slots necessary for P_i to finish. Moreover, after exceeding its time slot P_i is interrupted by all other slots for a constant period of time. With the following equation we can compute the exact value for the TDMA best case response time of a task:

$$r_i = c_i + \left(\left\lceil \frac{c_i}{t_{slot,i}} \right\rceil - 1\right) \cdot \sum_{(\forall) j \neq i} t_{slot,j} \quad (3)$$

c_i represents the minimum core execution time of P_i and $t_{slot,i}$ is the time slot of P_i.

We apply the above proposed approaches to the example system proposed in Section 3 and we obtain the results shown in Table 4.

Table 4. Analysis results applying the proposed best case algorithm

	Best-Case Approximation				Best Case = 0	Output Jitter
Task	Response Time	Input Jitter	Internal Jitter	Output Jitter	Output Jitter	Reduction (%)
P_1	[20;40]	54	20	74	172	56,9
P_2	[40;130]	0	90	90	170	47,1
C_1	[42;96]	0	54	54	96	43,7
C_2	[70;164]	74	94	168	429	60,8
C_3	[47;196]	90	149	239	426	43,8
P_3	[10;50]	168	40	208	554	62,4
P_4	[40;180]	239	140	379	762	50,2
					Average	52,1

If we again analyze the timing for the path IP2-P_2-C_3-P_4-HW2 we observe that the jitter corresponding to the event stream entering HW2 decreased by 50% when compared with the value obtained by considering best case equal zero. The burst size corresponding to this event model is reduced from 6 to 3.

Comparing the response time values from Table 2, 3 and 4 we observe that not only the minimum response times changed but also the worst-case response time values. This is because a better estimation of the best-case response time leads to a reduction of the output jitter, and thus, reducing the uncertainty at the input of the connected component.

Considering the best-case response time equal to minimum core execution time we obtain an average timing improvement of 28%. Applying the new algorithms to determine the minimum response time leads to an average optimization of 52% of the jitter values. This would allow a further reduction of the CPU speed or the speed of some other component, e. g. the bus. Again, the overall system cost can be lowered.

5. Other Task Configurations

In the preceding sections, we focused on a relatively simple task configuration, i. e. periodically activated tasks –possibly with jitter– with unconditional communication behavior. In this section, we extend the basic ideas to the other task configurations, i. e. tasks which are activated sporadically and/or have conditional communication behavior.

First, we consider tasks that are periodically activated but have conditional output behavior, i. e. they do not always produce an output (event) at each execution. In other words, the output stream is *sporadic*. The events timing in a sporadic stream is far less predictable than in a periodic stream because the periodic nature of input events and task execution is lost at the output. However, we can bound the event timing in the worst case and in the best case.

Audsley [1], already analyzed the properties of sporadic streams and came to a surprisingly simple result: in the worst case, which we consider the maximum number of events, in turn leading to a maximum load, a sporadic stream behaves exactly the same way as the corresponding periodic stream. In other words, a periodic stream model tightly bounds the worst-case behavior of a sporadic stream. This means, that we can directly reuse the output event models of periodic, unconditional tasks, as introduced above, as the worst-case bound for periodic but conditional tasks. The same response time calculations can be used.

Obviously, the best-case bound fundamentally differs. A best-case situation is said to have as few events as possible. A conditional task does not always produce an output on each execution. Hence, in a best-case output scenario, the task will not produce an output on any execution. In other words, it does not produce any output at all; the number of events is zero.

Such sporadic event models are known as input event models from scheduling analysis. The model of sporadic events basically provides two bounds, one for the worst and the other for the best case. But only the periodic worst-case bound is provided explicitly. The best-case bound is implicit, namely zero. This is indicated by the fact that the stream is considered *sporadic* instead of *periodic*.

We have not yet fully considered the detailed influence of scheduling. As explained in Section 2, a task execution can add jitter characteristics to an output stream. The model of periodic events with jitter is well known in scheduling analysis theory. However, there is no model of *sporadic events with jitter*, although it is now clear that such models make ultimate sense as output event models. One of the main reasons why sporadic events with jitter have not yet been considered is that output models in general have not yet received much attention. In [8], we introduced a six-class event model set that slightly extends the known models to capture the consequences of jitters, and subsequently bursts, in periodic as well as sporadic models, and we have shown that existing analysis techniques can be adapted to these models easily. The six-class model set includes simple periodic, periodic with jitter, and periodic with burst, as well as the sporadic counterparts.

To complete the approach on output event model analysis, we will shortly explain the remaining task configurations. Another type of task configuration includes tasks that are activated sporadically and have a conditional or unconditional output behavior. The sporadic task activation leads to a sporadic execution behavior, and subsequently, to sporadic behavior at the output. However, the behavior of the sporadic input stream can be bounded, just as previously explained. We can again differentiate between a worst-case and a best-case situation, but this time, we need to consider this already at the task input.

In the output worst case, we assume worst-case input, i. e. periodic activation, and can apply the output model determination such as in the previous sections. In the best case, we assume zero activations, and clearly, there is no task execution and no output. Again, we obtain a sporadic output event stream, just as with *periodic but conditional* tasks. Interestingly, we cannot only reuse the calculation but we also obtain the same results.

The fact that a task can be periodically or sporadically activated clearly influences other tasks on the same CPU, or other transmissions on the same bus. Scheduling analysis must recognize this. Especially in static priority scheduling, the best-case number of preemptions can differ. A sporadic task might never be activated and will thus never preempt any other task. This reduces the optimization

potential introduced in Section 4.1 to some extend. However, it is still better to use the minimum core execution time as the best-case response time than zero. The TDMA response time bounds described in Section 4.2 are not affected.

6. Conclusion

In this paper, we have thoroughly investigated the importance of output event models in distributed system timing analysis. A key observation is that worst-case output event models strongly depend on best-case scheduling. However, both have received only little attention in the past, and only very conservative approximations can be found in literature.

Recent approaches rely on output event models to combine several local scheduling techniques into a system-level timing analysis model. And the output event model accuracy was shown to have a tremendous impact on the accuracy of the overall analysis result.

We presented two optimizations: a simple, intuitive one which allows to neglect scheduling influence but is still better than the existing work; and a more sophisticated and novel best-case scheduling analysis approach which increases the output event model accuracy further.

This increase in accuracy reveals significant, previously unknown performance reserves in the system which can now be exploited. In our experiments, we could show that the required performance of individual components can be heavily reduced without violating any timing constraints, allowing to select cheaper components. This cost reduction shows the practical consequences of more accurate output event models, and underlines the importance of best-case scheduling analysis.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Journal of Real-Time Systems*, 8(5):284–292, 1993.

[2] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

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

[4] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings Real-Time Systems Symposiom*, pages 201–209, 1990.

[5] 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.

[6] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Tenth International Symposium on Hardware/Software Codesign (CODES'02)*, Estes Park, Colorado, USA, May 2002.

[7] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4), Apr. 2003.

[8] 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, Dec. 2003.

[9] 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.

[10] F. Wolf. *Behavioral Intervals in Embedded Software – Timing and Power Analysis of Embedded Real-Time Software Processes*. Kluver Academic Publishers, Boston, 2002.