High Precision Response Time Analysis of Tasks with Precedence Chains

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
Mälardalen Real-Time Research Centre (MRTC)
Box 883, 721 23, Västerås, Sweden
saad.mubeen@mdh.se

***Abstract*—Response-Time Analysis (RTA) is a powerful, mature and well established schedulability analysis technique for real-time systems. In order to get better utilization of system resources, RTA should not overestimate the response time of tasks in the system. This paper addresses the problem of losing system wide information about precedence chains and overestimation found in response time of tasks when current RTA is applied to a system where precedence chain dependencies among tasks exist. We show that when there are precedence chains with one activating event in a real-time system, a task under analysis cannot experience Worst Case Execution Time of all interfering tasks at the same time when they all experience their maximum release jitter.**

Keywords-Response-time analysis; Fixed priority scheduling; Tasks with offsets; Real-time systems.

I. Introduction

Nowadays, the majority of electrical products are controlled by an embedded computing system. Often, these products interact with an environment in a timely manner i.e. the embedded system is a real-time system. For such a system, the desired and correct output is one which is logically correct as well as delivered within a specified time. Examples of real-time systems are found in many domains such as automotive, aerospace, robotics, medical equipments, production facilities etc. Some of the characteristics of these systems are mix functionality with diverse requirements on timing and dependability, operating in resource constrained environments and software reliability.

A large class of these embedded real-time systems is safety critical and this means that system failure can result in catastrophic consequences such as endangering human life or the environment. System providers of safety-critical systems are required to ensure that the system is safe. One important activity in order to establish the safety of a real-time system is to provide evidence that actions by the system will be provided in a timely manner (e.g. each action will be taken at a time that is appropriate to the environment of the system). Thus it has become extremely important for the system providers to predict the timing behavior of such systems.

In order to provide evidence that each action in the system will meet its deadline, *a priori* analysis techniques, also known as schedulability analysis techniques, have been developed by the research community. Tindell [1] developed schedulability analysis for tasks with offsets and it was further extended by Palencia and Gonzalez Harbour [2]. In [3], it is claimed that amongst the more traditional, analytical, schedulability analysis techniques, the response-time analysis of tasks with offsets (RTA) stands out as the prime candidate because of its better precision and ability to analyze quite complex system behaviors. In order to achieve better utilization of resources, the response time calculated by the RTA should not have overestimation.

In this paper we present a problem that we have identified applying RTA [1], [2] to tasks with precedence chains. The core of the problem is that when precedence chains among tasks are modeled there is pessimism in the response times. By assigning each task the local information of offset and release jitter the system wide information about precedence chains is lost. We illustrate this problem with an example in Section III. We show that while considering the precedence chain relations among tasks with one activating event, a task under analysis cannot experience Worst Case Execution Time (WCET) of all interfering tasks at the same time when all the interfering tasks experience their maximum release jitter delay. In order for RTA to be efficiently applicable to industry, this pessimism must clearly be addressed since precedence relations are a common type of inter-task dependency and are widely used in industrial systems [3].

The rest of the paper is organized as follow: Section II presents the existing RTA with offsets and the task model used in this work. Section III discusses the research problem. In Section IV we present a summary of work in progress.

II. Response Time Analysis

RTA [4], [5] is a powerful, mature and well established schedulability analysis technique. It is a method to calculate upper bounds on response times of tasks in real-time systems. In crux, RTA is used to perform a schedulability test which means it checks whether or not tasks in the system will satisfy their deadlines. RTA applies to systems where tasks are scheduled with respect to their priorities and which is the predominant scheduling technique used in real-time operating systems today [3].

Liu and Layland [6] provided theoretical foundation for analysis of fixed-priority scheduled systems. Since then schedulability analysis of fixed-priority preemptive tasks has been well developed. Joseph and Pandya published the first RTA [7] for the simple task model presented by Liu and Layland which assumes independent periodic tasks. Subsequently, the RTA has been applied and extended in a number of ways by the research community. Moreover a number of issues have been attacked and resolved by the real-time research community such as lifting independent task assumption, analysis of communication networks, analyzing distributed systems, modeling of operating systems over-

heads, reducing pessimism from traditional RTA, making RTA faster and tighter etc.

A. Task Model With Offsets

This section describes the task model which we will use in our work. This transactional task model, also known as a task model with offsets, was developed by [1] and extended by [2]. This task model can be viewed as state-of-the-art task model for RTA in a sense that it can handle many different and complex parameters with few constraints as well as applied for different system models. Parameters used to define the system are arbitrary deadlines, release jitter, temporal dependencies through offsets, access to shared resources etc. Furthermore this task model can be applied to systems in which tasks suspend themselves or generally to any system where tasks may have temporal dependencies (offsets) among them. We will use the system model that is formally described as follows:

$$\begin{aligned}\Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\} \\ \Gamma_i &:= \langle \{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i \rangle \\ \tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} \rangle\end{aligned}$$

The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i. In the case of sporadic events, T_i is the minimum inter-arrival time between two consecutive events. In this model we consider that the activating events are mutually independent i.e. phasing between them is arbitrary. There are $|\Gamma_i|$ tasks in a transaction Γ_i and each task may not be activated until a certain time, called *offset*, elapses after the arrival of the external event. By task activation we mean that the task is released for execution.

We use τ_{ij} to denote a task. The first subscript *i*, specifies the transaction to which this task belongs and the second subscript *j* denotes the number of the task within the transaction. A task, τ_{ij}, is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum release jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). In this task model, there are no restrictions placed on offset, deadline or jitter, i.e. they can each be either smaller or greater than the period.

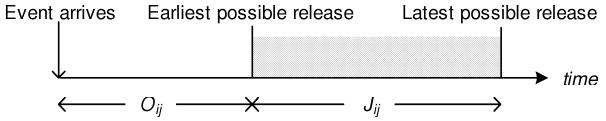

Figure 1. Relation between an event arrival, offset, jitter and task release.

The relation between event arrival, offset, jitter and task release is graphically depicted in Fig. 1. It is obvious from Fig. 1 that the offset (O_{ij}) corresponds to the earliest possible release of a task whereas the release jitter (J_{ij}) corresponds to latest possible release of the task. In other words, after the event arrival, task τ_{ij} is not released for execution until its offset (O_{ij}) has elapsed. The task release may be further delayed by release jitter (maximally until O_{ij} + J_{ij}) making its exact release uncertain. An example transaction (Γ_i) out of many transactions in a system is shown in Fig. 2. This transaction has two tasks i.e. τ_{i1} and τ_{i2}. The offset, release jitter, WCET of each task and period of the activating sequence of events are also shown in Fig. 2.

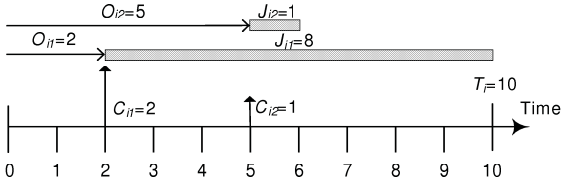

Figure 2. An example transaction Γ_i.

B. Distributed Systems with Precedences

The task model presented in the above subsection can be applied to e.g. distributed systems. Precedence relations among tasks naturally exist in distributed systems. Hence the original, and most widely adopted, application for task model with offsets is to model precedence relations among tasks in distributed systems [1], [2]. In this case, a transaction represents a group of tasks, allocated to several nodes, where every task has a precedence relation to previous task in the transaction. End-to-end response times are calculated over node and communication device boundaries by applying RTA to each node and also to the network to produce a holistic schedulability analysis.

The precedence relation in distributed real-time systems is modeled by means of dynamic *offset* [2] where the offset for a task represents the earliest possible release of a task based on how early the chain of preceding tasks are able to finish. The jitter term represents the latest possible release time, denoting the time instant where the chain of preceding tasks are able to finish i.e. the response time of the immediate predecessor of the task at hand.

Consider an example transaction with three tasks as shown in Fig. 3. Since task τ_{i1} is the first task to execute when the corresponding event arrives, its offset and release jitter is zero. The second task τ_{i2} is activated only when τ_{i1} completes. We assume that τ_{i1} will finish no sooner than 5 time units after the event arrival. This may be due to the interference from tasks belonging to other transactions in the system. Therefore, τ_{i2} will have an offset of 5 time units. The latest time τ_{i1} will finish is 7 which means that τ_{i2} will have a release jitter of (7-5) = 2 time units. Similar reasoning applies to τ_{i3}. The three tasks in Fig. 3 are shown in such a way that as if all of them suffer from their worst-case jitter.

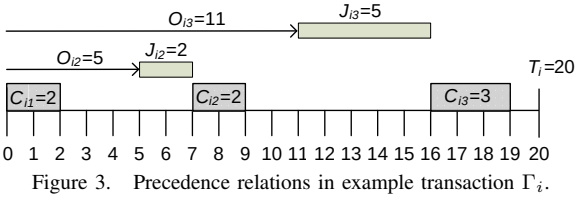

Figure 3. Precedence relations in example transaction Γ_i.

C. Hybrid Scheduled Systems

Precedence chain relations among tasks, with single activating event, also exist in hybrid scheduled systems. These

systems are called hybrid because they use both static and dynamic scheduling. In this subsection we describe the system model used in a commercial system [8] that supports hybrid scheduling [9]. In this model the dynamically scheduled tasks "run in the background" of a static schedule. The system model contains interrupts, static schedule and dynamic tasks. They can be modeled in any priority order.

- **Interrupts**. There may be multiple interrupt levels, i.e. an interrupt may be preempted by higher level interrupts. Each interrupt is modeled as a transaction with one single task having a period or minimum inter-arrival time.
- **A static cyclic schedule**. A schedule of a set of periodic tasks (functions), each task with a known WCET, is constructed off-line. The length (duration) of the schedule is equal to the least common multiple of all statically scheduled function periods. The static schedule is modeled as one transaction where offsets represent the time instant where tasks are released according to the static schedule. When the whole schedule has been executed the schedule is restarted from the beginning. It is assumed that the schedule is valid even if its functions are preempted by interrupts.
- **A set dynamically dispatched tasks**. These tasks are scheduled by a fixed priority preemptive scheduler. They are assumed to be periodic or to have known minimum inter-arrival time. They execute in the time slots available between interrupts and statically scheduled functions.

We illustrate this model with an example. Fig. 4 shows a static cyclic schedule of length 20, with 4 functions released at times 0, 5, 10 and 15 with WCETs 4, 1, 1 and 3 respectively. Thus this static cyclic schedule is modeled as a single transaction, activated at the start of the schedule, with four tasks having precedence chain relations shown from lower relaease time to higher as in Fig. 4.

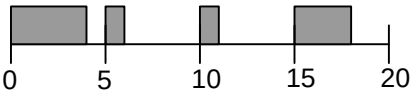

Figure 4. Example of static cyclic schedule

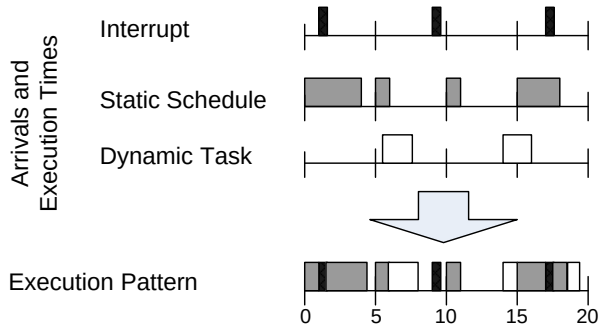

Figure 5. Example execution scenario

An example execution scenario is shown in Fig. 5 when schedule from Fig. 4 is executed with one interfering interrupt source and one dynamically scheduled task (two instances of that task are activated). In this example, we assume the following priority order:

$P_{(Interrupts)} > P_{(Static\ Schedule)} > P_{(Dynamic\ Tasks)}$

It is very important for such systems that the precedence chain relations are preserved.

III. Research Problem

A problem we have identified is that modeling and analysis of precedence chain relations among tasks, with single activating event, with current RTA results in the loss of system wide information about precedence chains and overestimation in calculated response-time. We illustrate this problem with an example in the next subsection.

A. Illustrative Example

Consider a transaction with three tasks i.e. τ_{i1}, τ_{i2} and τ_{i3}. These tasks have precedence relations among them such that τ_{i1} precedes τ_{i2} and τ_{i2} precedes τ_{i3}.

$$\Gamma_i := \langle \{\tau_{i1}, \tau_{i2}, \tau_{i3}\}, T_i = 100 \rangle$$
$$C_{i1} = C_{i2} = C_{i3} = 20$$
$$P_{i1} > P_{i2} > P_{i3}$$

Also consider a low priority task τ_{k1} with WCET, C_{k1} = 30. This will be the task under analysis and we are interested in calculating its worst-case response time. We assume that there are no more tasks in the system.

We further assume that the best-case execution time of these tasks is very small and hence, the jitter for a task is approximately equal to the response time of its predecessor. In other words, since jitter represents uncertainty in a task's release, the difference between best and worst-case response time of a preceding task becomes the successor's release jitter. Let us denote the response time of a task *j* belonging to transaction *i* by R_{ij}. Since task τ_{i1} is the highest priority task in the system, its release jitter is zero. Release jitter of the three higher priority tasks (compared to task under analysis) is given as follows:

$$J_{i1} = 0 \Rightarrow\ R_{i1} = 20 \Rightarrow$$
$$J_{i2} = 20 \Rightarrow\ R_{i2} = 40 \Rightarrow J_{i3} = 40$$

Hence, the release jitter of task τ_{i2} is inherited from the response time of task τ_{i1}. Similarly, the release jitter of task τ_{i3} is equal to the response time of task τ_{i2}. The release jitter term means that a task that is released periodically may sometimes be delayed at most with its jitter term. This has an impact when considering the response time of task τ_{k1} where the interference of each higher priority task is considered separately.

According to the RTA developed in [2], the maximum busy period for the low priority task τ_{k1} occurs when the activations of higher priority tasks occurring (i) before or at the critical instant are delayed by the amount of jitter such that they all occur at the critical instant and, (ii) after the critical instant experience zero jitter. Furthermore, the task of transaction Γ_i coinciding with the critical instant experiences its worst-case jitter delay. The critical instant is defined in [1]. Therefore, in this example, the worst case

interference τ_{i3} imposes on τ_{k1} is when it first suffers its worst-case jitter and subsequent releases occur with no jitter. The resulting scenario is shown in Fig. 6.

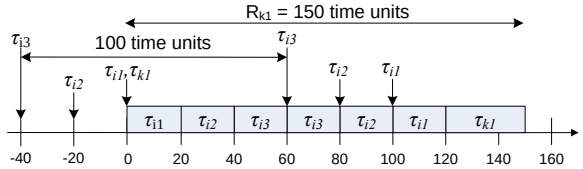

Figure 6. Correlated WCETs with current RTA

The above discussion implies that the worst-case interference occurs when a task is released , experiences its worst-case release jitter, coincides with the release of the task under analysis, and finally is released again after period time units (100 in this example) has elapsed from the previous release. This fact is highlighted for τ_{i3} in Fig. 6. Similar reasoning applies to τ_{i2}. By applying RTA [1], [2] the response time of task τ_{k1} i.e. R_{k1} becomes 150 time units as shown in Fig. 6. The reason for this is that it is assumed that all tasks experience their worst-case jitter and WCET simultaneously and independently. Moreover, it is also evident from the figure that the precedence chain relations among tasks are not preserved.

However, in order for the task τ_{k1} to really experience its worst-case release jitter, τ_{i2} and τ_{i3} would have to execute for almost zero time units and hence τ_{k1} cannot experience both WCET of all three tasks and at the same time they all experience their maximum release jitter delay. In reality the task τ_{k1} can only be interfered by each task at most once and hence the worst-case response time of task τ_{k1} in reality cannot exceed 90 time units as shown in Fig. 7. The figure also shows that the information of precedence chain relations among tasks is also preserved.

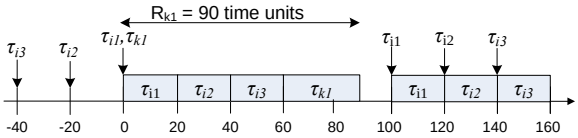

Figure 7. Correlated WCETs without overestimated RTA

B. Pessimistic Modeling and Analysis of Tasks with Precedence Relations

We see from the above example that when current RTA is applied to task with precedence chain relations among them, with single activating event, there is pessimism in the calculated response- time. The essence of the problem is that precedence chain relations are modeled with jitter and in RTA each task has its own local jitter without preserving information about precedence chains. We show that while considering such systems, a task under analysis cannot experience WCET of all interfering tasks at the same time when all the interfering tasks experience their maximum release jitter delay. Moreover, when systems with such task models are analyzed then system wide information on precedence chains is lost.

It should be noted that this problem is not as apparent in distributed systems as in single processor systems. As an example, consider a distributed system in which each task runs on a separate processor and the tasks have precedence chain relations among them. Hence, it is possible that a task under analysis can experience WCET and worst-case jitter from all interfering tasks simultaneously. But the precedence chain relations among the tasks should be preserved. We identified this problem only in single processor systems having tasks with precedence chain relations, with single activating event, as discussed in Section II-C. In order for RTA to be efficiently applicable in commercial systems, this pessimism must clearly be addressed since precedence chain relations are a common type of inter-task dependency and are widely used in industrial systems. Therefore, there is a need to develop high precision RTA by extending the existing RTA [1], [2] for real-time systems having tasks with precedence chains.

IV. Summary

In this paper we presented the problem of overestimation in the calculated response-time and loss of precedence chain information among tasks when RTA is applied to tasks with precedence chains. We illustrated this problem with an example. We also discussed the systems in which precedence chains among tasks exist and which can be affected by this problem. Moreover, we have shown that while considering the precedence chain relations among tasks, a task under analysis cannot experience WCET of all interfering tasks at the same time when all the interfering tasks experience their maximum release jitter delay. Currently we are working on this problem and trying to extend RTA to enable it to analyze such systems without overestimation while preserving the precedence chain relations.

Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF.

References

[1] K. W. Tindell, "Using offset information to analyse static priority preemptively scheduled task sets," Dept. of Computer Science, University of York, Tech. Rep. YCS 182, 1992.

[2] J. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," *Real-Time Systems Symposium, IEEE International*, vol. 0, p. 26, 1998.

[3] M. Nolin, J. Mäki-Turja, and K. Hänninen, "Achieving industrial strength timing predictions of embedded system behavior," in *ESA*, 2008, pp. 173–178.

[4] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling:an historic perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 173–198, 1995.

[5] L. Sha, T. Abdelzaher, K.-E. A. rzén, A. Cervin, T. P. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-Time Systems*, vol. 28, no. 2/3, pp. 101–155, November/December 2004.

[6] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[7] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal (British Computer Society)*, vol. 29, no. 5, pp. 390–395, October 1986.

[8] "Arcticus Systems," http://www.arcticussystems.com.

[9] J. Mäki-Turja, K. Hänninen, and M. Nolin, "Efficient development of real-time systems using hybrid scheduling," in *International Conference on Embedded Systems and Applications*, June 2005.