Towards Response-Time Analysis of Complex Real-Time Systems by using Parametric Worst-Case Execution-Time Estimate on Tasks – A Case Study for Robotic Control System

Yue Lu, Antonio Cicchetti, Mikael Sjödin, Jukka Mäki-Turja, Stefan Bygde and Christer Norström
Mälardalen Real-Time Research Centre
Mälardalen University, Västerås, Sweden
{yue.lu, antonio.cicchetti, mikael.sjodin, jukka.maki-turja, stefan.bygde, christer.norstrom}@mdh.se

Abstract

Avoiding timing-related errors in complex industrial real-time software systems becomes more and more important. In our target domain, such complex software systems are developed in C and realized with periodic tasks, executed on a single processor under fixed-priority preemptive scheduling. The tasks exhibit intrinsic dependencies with respect to execution times; often the execution time of a task is dependent from inter-process communication or globally shared variables. Thus, the execution time is highly variable and using a pessimistic, safe, upper bound on the execution time (worst-case execution time, WCET) in analysis of the system would give unacceptable pessimistic results. However, existing techniques for schedulability analysis, such as the well known Response-Time Analysis (RTA), typically use WCET as their input.

In this paper, we illustrate the problem by using a timing model inspired by a robotic control system from ABB. We show how models of tasks whose execution time is dependent on asynchronous message-passing and globally shared state variables, and how we can model this execution time as a parametric WCET (PWCET). Further, we show how we, in this example can use TIMES to formally derive the parameters for the PWCET in order to obtain a concrete WCET to be used in the RTA.

1 Introduction

Most existing embedded real-time software systems today have been developed in a traditional code-oriented manner. Many such systems are maintained over extended periods of time, sometimes spanning decades, during which the systems become larger and increasingly complex. The result is that these systems are difficult and expensive to maintain and verify. In our target domain, robotic control systems, such complex systems often consist of millions of lines of C code.

The typical software architecture is based on periodic tasks, executed on a single processor under Fixed-Priority Preemptive Scheduling (FPPS) and stringent real-time constraints. However, contrary to the assumption in most real-time theory, the tasks exhibit strong dependencies between them. Often, the execution time of a task is heavily dependent on data generated by other tasks, such as the number of messages sent to a task via Inter-Process Communication (IPC) primitives, or data placed in globally shared variables and buffers.

One specific problem when maintaining such complex systems is the risk for introducing timing-related errors. A desirable approach to avoid timing-related errors in real-time software would be to use schedulability analysis methods, such as Response-Time Analysis (RTA), that provide information on timing behavior of execution in worst-case scenarios. However, RTA (and other schedulability analysis techniques) use the Worst-Case Execution Time (WCET) of tasks as input. The quality of the analysis is directly correlated to the quality of the WCET estimates. Hence, using overly pessimistic WCET estimates would yield unacceptable pessimistic RTA results.

Unfortunately, the above described systems may not have any easy computable WCET. Sometimes a pessimistic WCET bound can be calculated based on maximum queue or buffer lengths. In contrast, in other cases the WCET is completely unbounded before we know the behavior of dependent tasks. Let us consider the following simplified example in Figure 1, taken from an industrial robotic control system, where a task reads all messages buffered in a message queue and processes them accordingly: to find the exact number of messages actually consumed in the worst-case scenario by inspecting the code is very hard since other tasks (with more significant priority) may preempt the execution of the loop and refill the queue at runtime. Moreover,

```
msg = recvMessage(MyMessageQueue);
while (msg != NO_MESSAGE) {
  process_msg(msg);
  msg = recvMessage(MyMessageQueue);
}
```

Figure 1. Iteration-loop wrt. message passing

this worst-case behavior is also nonintuitive: the number of loop iterations is not supposed to be bounded by the maximum queue size when preemption occurs. Surprisingly, our evaluation work presented later in Section 4 shows a different result, i.e., such number is bounded and even smaller than the maximum queue size.

This work relies on the observation that many of the tasks in these systems can be characterized by a Parametric WCET (PWCET) (we will later show an example in Section 3), where, for instance, the PWCET is a function of the number of messages a task may need to handle during one invocation. We show how to calculate response time of such a task with PWCET, based on models of the other tasks in the system. In order to have a better understanding, we use a concrete example inspired by a control system for industrial robots developed by ABB.

2 System modeling of complex real-time system

Our target system, i.e., a robotic control system, is quite large, and contains around 3 millions lines of code. In order to construct PWCET estimates on each task in the system, a system model which describes the detailed execution control flow on a code level is needed. In particular, information with respect to resource usage and interaction, e.g., inter-process communication (IPC), CPU time and logical resources., can be extracted from the real system based on program slicing. In this work, such model is described by the modeling language used in RTSSim, which can be considered as a domain-specific language describing both architecture and behavior of task-oriented systems, running on a single processor and developed in C. Its syntax and semantics are as expressive as C programming language, but are extended with typical RTOS services to the task models, such as task scheduling (e.g. FPPS), IPC via message passing[1] and synchronization (semaphore), for instance. For the purpose of this work, it is sufficient to know that RTSSim employs a hierarchical model to specify the system structure consisting of a number of *tasks*[2]. A task may not be released for execution until a certain time, called *offset*, has elapsed after the arrival of the activating event. Each task is characterized by a *period*, a *maximum jitter*, and a *priority*. Periods and priorities can be changed at any time by any task in the application, the lower numbered priorities are more significant. Finally, each task is composed of a number of *jobs* and invoked RTOS services. Each job in RTSSim task is represented by the modeling primitive *execute* based on static WCET estimate. E.g., `execute(tcb, 100, 10)` means the adhering task will consume 10 model-time units with 100 percent possibility.

[1]Refer to code line 25 and 38 in Figure 2 as the examples.

[2]We intentionally missed out some details of the language for the sake of space. The interested reader is referred to [1] for a thorough description of RTSSim

3 RTA using PWCET

Model 1, as described in this section, is designed to include some behavioral mechanisms from the robotic control system, which make to predict tasks' WCET estimates very difficult. Moreover, it is used as a concrete example to illustrate the proposed way of modeling tasks' execution time as PWCETs.

Model 1 contains a First-In-First-Out (FIFO) buffer IOQ with queue size 13 and three periodic tasks i.e., ENV_IO, IO and CTRL task with the parameters shown in Table 1 (a lower valued priority is more significant). The scheduling policy in Model 1 is FPPS, and all the tasks are running on a single processor. ENV_IO task is an environmental task which generates 2 external events that are stored in the globally shared variable `nofEvents` as shown in code line 7 in Figure 2. IO and CTRL tasks have a complex temporal behavior due to input-dependent data placed in queue IOQ and to the globally shared variable `nofEvents`, which varies their execution times radically:

1. IO task sends an uncertain number (from 0 to 6) of messages to queue IOQ depending on the value of the global variable `nofEvents` : if the value is bigger than 6, then there will be at most 6 messages sent to queue IOQ;otherwise, the number of messages sent by IO task is the same as the value of `nofEvents`. Once a message is sent to queue IOQ, the value of `nofEvents` is accordingly decreased by 1, in order to denote that one external event is successfully stored in queue IOQ. Moreover, the number of messages sent by IO task is counted by the local variable `k` (see 15-27 lines of code in Figure 2);
2. When CTRL task starts running, it will at first consume all the messages stored in queue IOQ. However, due to its lower priority, CTRL task may be preempted in the loop `do while` by IO task, which refills an uncertain number of messages in the way described previously. This will increase the number of messages consumed by CTRL task, which is counted by the local variable `i` (see 37-44 lines in Figure 2). After consuming all the

messages in queue IOQ, CTRL task may continue its execution with 10 model-time units more and increase the value of the local variable `j`, depending on whether the value of the globally shared variable `nofEvents` is bigger than 5 (lines 46-52 in Figure 2).

Table 1. Task parameters for Model 1

Task	*Priority*	*Period*	*Parameters*
ENV_IO	0	200	No
IO	1	500	`k`
CTRL	2	800	`i,j`

Clearly in Model 1, the value of the local variables `k`, `i` and `j` are vital to calculate the WCET estimates on the adhering tasks since they count the number of times of primitives (e.g., `sendMessage(tcb, IOQ, 1, 0)`, `recvMessage(tcb, IOQ, 0)` and `execute(tcb, 100, 10)` being executed in basic control structures such as loop `while`, `do while`, `for`, and `if/else` statement. However, the complexity of obtaining the exact value of such local variables is very high, e.g., the value of `i` is difficult to predict since the number of messages consumed by CTRL task when preemptions occur may not be bounded by the maximum queue size. Even worse, to trace the value of the globally shared variable `nofEvents` along with tasks' execution is difficult since all of the three tasks have the dependency with it. Adopting non-parametric WCET analysis to estimate WCET on tasks in Model 1 cannot capture such intricate dependencies. Whereas, if variables `k`, `i` and `j` are considered as *parameters*, tasks' intricate dependencies through data placed in globally shared variables and queues can be represented in PWCET formula for each task. By using these PWCET formula in RTA, such complex dependencies are preserved, which discloses the possibility to perform optimistic and safe RTA. By inspecting code in Model 1, the PWCET estimate on IO and CTRL task can be expressed as two following linear equations, respectively, which are function of parameters (or local variables) `k`, `i` and `j`:

Equation 1. $C(IO) = k \times C(sendMessge)$

Equation 2. $C(CTRL) = i \times C(recvMessage) + 2 + j \times C(10)$

where $C(sendMessage)$ and $C(recvMessage)$ are 2 model-time units consumed by primitives `sendMessage(tcb, IOQ, 1, 0)` and `recvMessage(tcb, IOQ, 0)`; $C(10)$ is 10 model-time units consumed by primitive `execute(tcb, 100, 10)`. Once the values of the parameters are obtained, the WCET estimate on each task will be used in Equation 3 in RTA .

```
1  #define IOQSIZE 13
2
3  int nofEvents = 0;
4
5  void RTSSim_ENV_IO(TCB* tcb)
6  {
7     nofEvents += 2;
8  }
9
10 void RTSSim_IO(TCB* tcb)
11 {
12   int eventsToProcess = 0;
13   int k = 0;
14
15   if (nofEvents > 6)
16   {
17     eventsToProcess = 6;
18   }else{
19     eventsToProcess = nofEvents;
20   }
21
22   while(eventsToProcess-- > 0)
23   {
24     nofEvents--;
25     sendMessage(tcb, IOQ, 1, 0);
26     k++;
27   }
28 }
29
30 void RTSSim_CTRL(TCB* tcb)
31 {
32   int sc_var1 = 0;
33   int ioevent = 0;
34   int i = 0;
35   int j = 0;
36
37   do{
38     ioevent = recvMessage(tcb, IOQ, 0);
39
40     if (ioevent > -1)
41     {
42       i++;
43     }
44   }while (ioevent > -1);
45
46   sc_var1 = nofEvents;
47
48   if(sc_var1 > 5)
49   {
50       execute(tcb, 100, 10);
51       j++;
52   }
53
54 }
```

Figure 2. An RTSSim model case study for method illustration

Equation 3. $R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j$

where the set of values $\{R^0, R^1, R^2, R^3, ...\}$ is monotonically non-decreasing.

In order to ensure the exact value of parameters can be derived, formal analysis such as model checking will be used, which is presented in the following Section 4.

4 Obtaining parameter values

In the following, we propose an approach to formally derive the value of parameters in PWCET formula by relying on a concrete *semantic anchoring* process as presented in [2]. In particular, Model 1 is firstly transformed to a target representation, which is formally analyzable by the model checking tool TIMES [3]. Such a translation is performed at the meta-model level, thus allowing to preserve the original semantic[3]. Thereby, the exact value of parameters can be ensured to be derived by analyzing all the possible values in the entire search space[4]. Due to page limit, we won't introduce the rules used for such transformation, since the details of such transformation go beyond the scope of this work. In contrast, we show how the parameters look like after the transformation by referring to Figure 3, which shows the translation of CTRL task (from code line 32 to 52 in Figure 2) to its counterpart, i.e., a task automata model in TIMES. For instance, the counterpart of the parameter `i` in CTRL task is a bounded integer `recvIOQ_i_ctrl` in TIMES model preserving the same semantics, that is, it counts for the number of times of primitive `recvMessage` being executed.

Then, by manually checking the *reachability* property wrt. `recvIOQ_i_ctrl`, such as *E<> recvIOQ_i_ctrl == 11* (with verification result `Not satisfied`) and *E<> recvIOQ_i_ctrl == 10* (with verification result `Satisfied`) in TIMES, the exact value of parameter `i` is found to be 10. This highlights an interesting point that the number of loop iterations counted by the parameter (or the local variable) `i` is bounded and even smaller than the maximum queue size when preemption occurs, although intuitively it is not supposed to be like that.

[3]The RTSSim model and its counterpart in TIMES describe the same system behavior and valued data.

[4]This is under the assumption that the entire search space is reasonably big as not raising state space explosion issues.

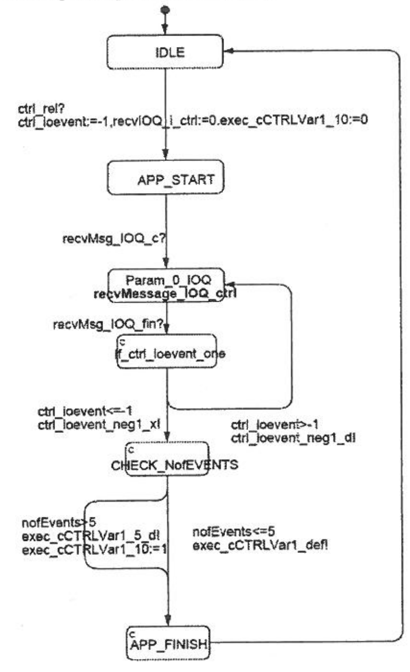

Figure 3. tat_tat for CTRL task (from code line 32 to 52 in Figure 2), after semantic anchoring

5 Conclusions and future work

In this paper we present an idea about using Parametric Worst-Case Execution-Time (PWCET) estimate on tasks in Response-Time Analysis of complex real-time systems. By using a concrete example inspired by ABB robotic control system, tasks intricate dependencies are firstly introduced. In particular, input-dependent data placed in globally shared variables and buffers, which vary tasks' execution time radically, have been highlighted. Then, the way of modeling such tasks' execution time as PWCET centre around a number of parameters, is proposed. Later, the way of formally deriving the value of parameters in PWCET by using model checking tool TIMES is illustrated.

As part of the future work, we will validate our idea by evaluating more complex models, by for instance increasing the number of parameters, tasks and queued buffers. In this respect, the possibility of using alternative analysis wrt. obtaining the value of parameters will be explored. More importantly, we will pursue the generalization of the current proposal to different complex legacy system modeling approaches, and possibly different formal translations to support multiple forms of analysis.

References

[1] Johan Kraft. Rtssim - a simulation framework for complex embedded systems. Technical Report, Mälardalen University, March 2009.

[2] Yue Lu, Antonio Cicchetti, Stefan Bygde, Johan Kraft, Thomas Nolte, and Christer Norström. Transformational specification of complex legacy real-time systems via semantic anchoring. In *2nd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS 2009) @ COMPSAC*. IEEE Computer Society Press, July 2009.

[3] Website of times. www.timestool.com.