Probabilistic Timing Analysis of a Periodic Task on a Microcontroller

Jonathan Thörn[1], Najda Vidimlic[1], Anna Friebe[1,2], Alessandro V. Papadopoulos[1], Thomas Nolte[1]
[1]Mälardalen University, Västerås, Sweden
[2]Åland University of Applied Sciences, Mariehamn, Åland, Finland

***Abstract*—In this paper we present our ongoing work towards a realistic probabilistic timing analysis of embedded software systems subject to timing requirements. In order to provide such an analysis that captures necessary and important behavioural features of the software system under analysis, including the underlying platform, we have implemented a real-time system running on a Rasberry Pi microcontroller on which we have performed a series of experiments and measurements. The results so far suggest a new model for analysis that captures more detailed behaviour and consequently provides a more accurate and correct probabilistic analysis.**

I. Introduction

In the context of embedded software implementing hard real-time systems, traditional scheduling algorithms and analysis focus on strict deadlines and deterministic timing guarantees that must be satisfied [1]. Instrumental in realizing such analyses and scheduling algorithms is knowledge concerning software execution time and algorithm behaviour. In particular, Worst-Case Execution Times (WCET) and Worst-Case Response Times (WCRT), which both often are pessimistic by design. This led to different research directions in the real-time community, investigating different approaches to reduce such pessimism in practice, such as the Typical Worst-Case Analysis (TWCA) [2], or probabilistic approaches [3]–[7]. However, in soft real-time systems it is sufficient to look at probabilistic guarantees, i.e., there are requirements on the time distributions' tails that need to be fulfilled. These requirements differ between systems. Commonly the likelihood of a deadline miss needs to be below a given threshold and/or the amount by which a deadline is missed needs to be within certain bounds. When designing a soft real-time system based on worst-case timing analyses, the resulting system will be over-provisioned. This causes increased costs and additional resources such as material and power consumption. Probabilistic (soft) real-time systems, on the other hand, have a potential to mitigate this over-provisioning of resources.

In this paper we present our ongoing work towards a more realistic, compared to current state-of-the-art, probabilistic modelling and analysis of real-time systems. In doing so we utilize embedded hardware where we set up controlled experiments to derive realistic models and analysis. Microcontrollers such as Raspberry Pi are commonly used in soft real-time systems. Such systems can be found in, e.g., robotics and Internet-of-Things (IoT) applications. In such applications, soft real-time systems need to have a predictable timing behaviour, i.e., we need to show that the probabilistic timing guarantees are fulfilled.

Several solutions to stochastic analysis methods have been developed over the past decades, for example in [6] a method for analysis of periodic real-time tasks is proposed. To be able to apply stochastic analysis methods for real systems, we need knowledge of the properties of the systems' time distributions. Moreover, we need knowledge and solutions that capture or correctly deal with potential dependencies among software and/or hardware. To the best of our knowledge the current state-of-the-art has limited or no support of such dependencies.

In this paper, we describe on-going work in investigating the response time distribution properties of a periodic task running on a Raspberry Pi microcontroller. In particular, effects on response times of interference from other processes are investigated. The release time distribution is modelled as a Markov chain.

The paper is structured as follows: after the introduction of Section I, used concepts and definitions are described in Section II. Next, the experimental setup is clarified in Section III and used methods are described in Section IV. Further, results are presented in Section V. Finally, the results are summarized and future work directions are discussed in Section VI.

II. Concepts and definitions

In this section the concepts used throughout the paper are defined to avoid misinterpretations.

Response Time: For each instance of a task, the response time is the time span from the release time (sched_wakeup event of the instance) until completion of execution.

Execution time: For each instance of a task, the execution time is the total time that a job is *executing* from when the job's execution starts until the execution is completed.

Wake-up latency: For each instance of a task, the wake-up latency is the time span from the release time (sched_wakeup event of the instance) until the scheduling time, when a CPU context switch enables the job to begin execution.

Interference: An instance of a task is subject to interference if executions by other processes on the CPU occur during the response time of the job.

This work was partially supported by the Swedish Research Council (VR) for the project "Practical Probabilistic Timing Analysis of Real-Time Systems (PARIS)".

Independent task(s): Tasks are considered independent (in this paper) since they do not share any software resources (e.g semaphores) among each other.

Time units used in graphs and calculations: For calculating and presenting response time, execution time and latency, all time references are given in microseconds.

Instance trace graph: This graph represents the response times for the job in each period $[0, n]$, where n is the number of periods in the referred recording session. This graph is used to analyze dependencies between succeeding task instances and the potential effect on response time.

Response time distribution graph. This graph is a histogram, where job response times are sectioned into intervals with a difference of one microsecond. This gives an estimate of the empirical probability distribution for the response times.

III. Experimental setup

In this section the foundation for the experiments is described. This includes the hardware setup, the test software and the data collection method chosen.

A. Hardware and configurations

The experiments are performed on a Raspberry Pi 3B[1] with a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU. To achieve real-time properties the Raspbian kernel is patched with PreemptRT, version 4.14.52. For deterministic execution the clock frequency is pinned to the turbo-mode, which gives a constant 1.2GHz. Before starting experiments this is verified by a simple batch script which prints the current frequency.

B. Software implementation

A simple program, written in C, is used for the tests. The program implements a single independent thread, utilizing the `pthread` library which is specified by the IEEE POSIX 1003.1c standard. The program simulates workload with predefined execution time, achieved by a loop of arithmetic computations. In order to ensure consistent execution the program is assigned the highest possible real-time priority and is scheduled according to FIFO. By setting affinity the program is pinned to CPU3, simplifying recordings and eliminating migration between cores. The program is implemented as a periodic task with absolute period of 5ms, achieved with continuous timestamps regulating the `clock_nanosleep()` function with `CLOCK_MONOTONIC` as reference.

C. Data collection

The command-line tool `trace-cmd`[2] is used to access `ftrace`[3] and to record data from the kernel during execution of the program. `ftrace` is a kernel tracer in Linux that can trace most functions during execution on the kernel. For execution trace visualization, Kernelshark[4] which is a GUI to `trace-cmd`, is used.

[1]https://www.raspberrypi.org/products/raspberry-pi-3-model-b

[2]https://lwn.net/Articles/410200

[3]https://elinux.org/Ftrace

[4]https://lwn.net/Articles/425583/

IV. Methodology

In this section the method for collecting and processing data is described. Further, we clarify the features of the timing distributions that we investigate along with the utilized methods.

A. Data acquisition and preprocessing

While the test program runs, a `trace-cmd` command is executed to record all events of interest. Several different sessions with varying recording time are conducted in the interval from 5 minutes to approximately 1 hour of continuous execution. This is done in an attempt to observe differences in response time in relation to the number of executed task instances.

After each recording session the log file is converted to a text file to extract relevant data for timing calculations. A Python script is used to read the text file, sort out events of interest, save these and perform calculations. The extracted events are `sched_wakeup` and `sched_switch`. These events are used for calculating the response time, execution time and wake-up latency. The results are exported for further analysis.

B. Investigated timing distribution features

In the python script mentioned above, calculations are based on the pattern of events in the log file.

The event pattern for a task is a repetition of: wakeup, switch in, and switch out. This is used to calculate the following features, where n is the instance of interest:

- Wake-up latency = $switch(in)_n - wakeup_n$
- Execution time = $switch(out)_n - switch(in)_n$
- Response time = $switch(out)_n - wakeup_n$

The execution time can be calculated in this manner because there are no preemptions of the process in the investigated logs, i.e, in each period there is exactly one time at which the process is switched in and one at which it is switched out. The response time distribution is visualized by displaying it as a histogram. Sequential dependencies of the response times are visualized using the instance trace graph. The relation between response times and wake-up latencies are investigated, in particular the increases in response time and wake-up latency for jobs affected by interfering processes. This relation is visualized in a sequential section of the recorded data. The logs and the code used for analysis are available online[5].

V. Results obtained so far

In this section findings derived from the experiments are presented.

A. Response time distribution

The collected data is analyzed in MATLAB and initially displayed as a histogram. Several different data sets originating from different recording sessions are analyzed, all indicating similar distributions. No differences are observed between

[5]https://gitlab.com/najda.vidimlic/timing-analysis

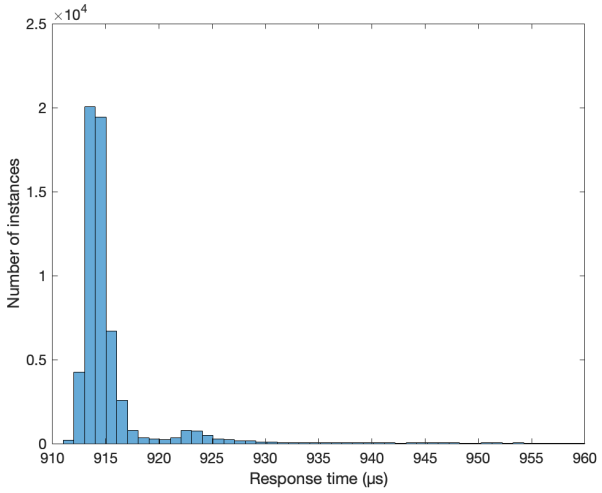

Fig. 1. Response time distribution.

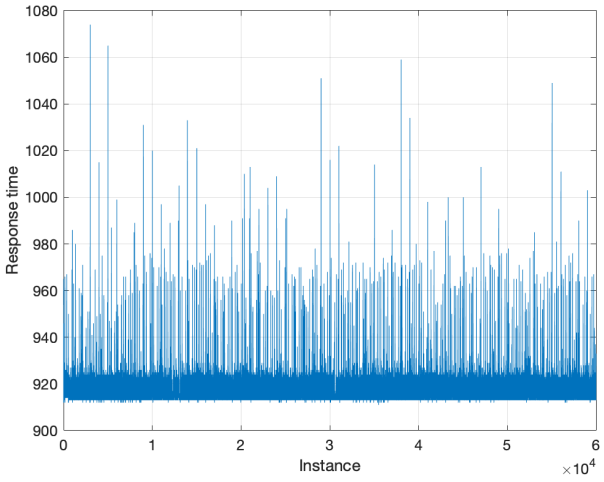

Fig. 2. Instance trace graph for 5 min recording session.

the different recording sessions and thus no further attempt is made in distinguishing differences originating from the number of executed task instances.

Fig. 1 shows the response time distribution for a recording session of 5 minutes. The distribution resembles a bimodal distribution, due to an initial large peak in the frequencies of the response time, and a second smaller peak around 923μs. A smaller number of jobs have notably longer response times and cause a local maximum of the empirical probability distribution.

a) Trace of response time in relation to instance: A bimodality in the distribution is observed, and the data is presented in the instance trace graph in a attempt to understand the reason for the increased response times causing the second peak. As seen in Fig. 2, several spikes in the response time are occurring at a seemingly periodic rate. To understand the cause of these spikes, additional information from the data sets is analyzed.

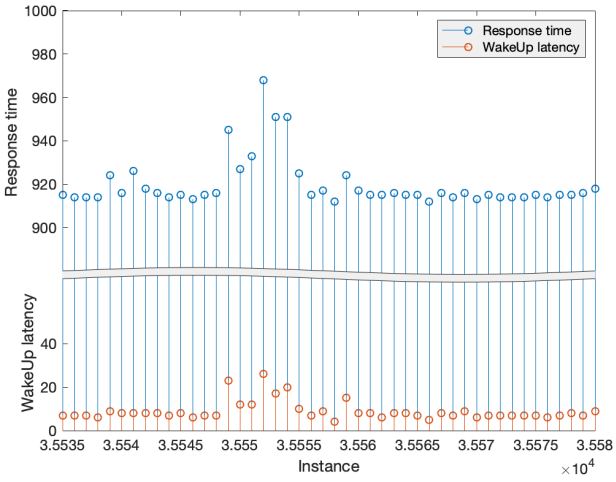

Fig. 3. Showing correlation between response time and wake-up latency.

b) Response time and latency correlation: Both the response time distribution and instance trace graphs indicate that some interference may be causing the increased response time. In Fig. 3, an instance trace graph consisting of both response time and wake-up latency data is shown. It could be expected that an increase in wake-up latency would give an equal increase in response time. However, this does not seem to be the case. The increase in response time is generally significantly greater compared to the increase in wake-up latency.

c) Response time distribution with interference: The discrepancy between latency and response time gives reason for additional analysis in search for factors contributing to the anomalies. The `trace-cmd` command is thus extended to record all events occurring during run-time. Hence new data sets are used from this point forward. An analysis of the collected data show additional sources of interference. These are grouped into different categories depending on their origin and occurrence:

- **RT-NoInter:** All task instances that have no interference.
- **RT-InterKTS-Exe:** All instances where the process `ktimersoftd` is interfering during execution.
- **RT-InterKTSandSIRQ-Exe:** All instances where both the processes `ktimersoftd` and `softirq` are interfering during execution.
- **RT-InterKTS-WakeUp:** All instances where the process `ktimersoftd` is interfering during wake-up latency.
- **RT-SwitchToKTS:** All instances where a context switch to the process `ktimersoftd` is performed after completing execution.

The interference in the analyzed logs consists of release (wake-up) events from different processes.

The distribution graph in Fig. 4 shows that the instances contributing to the second mode of the distribution mainly

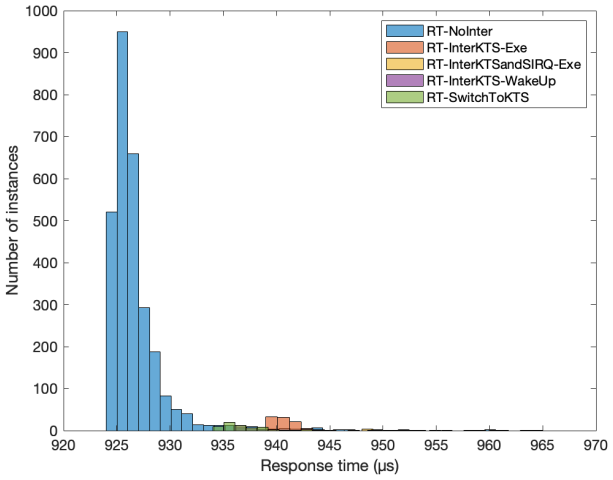

Fig. 4. Cause of prolonged response time identified to some extent.

consist of jobs affected by the different types of interference listed above.

B. Markov Analysis

From the graph in Fig. 4 two states can be identified: (i) execution without interference, and (ii) execution with interference. The system is modelled as a Markov chain with these two states. To approximate the probability of being in either of the states and transitioning from one state to another a simple Markov analysis is performed using the same set of data that is used to construct the graph in Fig. 4.

The data set contains 2845 instances, sorted into categories of the two states. The transition probabilities are estimated based on the state transition count:

$$\begin{aligned} &NI \to NI : 2479, \\ &NI \to I : 183, \\ &I \to I : 0, \\ &I \to NI : 183. \end{aligned}$$

The probability of being in the NI state (p_{NI}) or in the I state (p_I), as well as the transition probabilities λ_i, $i = 1, \ldots, 4$, are then calculated as

$$\begin{aligned} P_{NI} &= \frac{2479 + 183}{2845} \approx 0.93577 \\ P_I &= \frac{183}{2845} \approx 0.064323 \\ \lambda_1 &= \frac{2479}{2479 + 183} \approx 0.9312547 \\ \lambda_2 &= \frac{183}{2479 + 183} \approx 0.0687453 \end{aligned}$$

Since no transitions exist where the state remains in interference $\lambda_3 = 0$ and $\lambda_4 = 1$. Fig. 5 shows the resulting Markov chain.

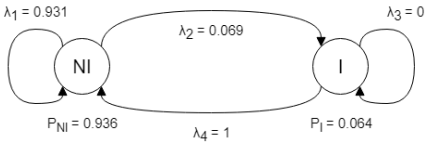

Fig. 5. Markov model of the two states and possible transitions.

VI. Summary and Future Work

Results obtained from the conducted experiments suggest that even though a task runs at highest priority and is not preempted the response time may change in ways that were not foreseen. The experiments also indicate that the response time has a bimodal distribution, regardless of quantity of traced instances. It has been observed that the second mode is to a large extent comprised of task instances with the identified interference types. The Markov model suggests a high probability of remaining in a non-interference state and also implies that consecutive states of interference are improbable. However, all anomalies in response time including some contained in the second mode can not be explained by the identified interference types.

Ongoing and future work will consist of further attempts to find, analyze and explain response time anomalies. Further, experiments with higher processor utilization could be conducted in order to elicit consecutive states of interference and effects on the distribution. Additionally, the state of interference in the Markov model could be partitioned into several sub-states for enhanced understanding of possible transitions. In addition, one to the above complementing and ongoing work aims at identifying probability distributions that can be used as tight upper bounds of the tail.

References

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

[2] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst, "Typical worst case response-time analysis and its use in automotive network design," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[3] L. Santinelli and L. Cucu-Grosjean, "Toward probabilistic real-time calculus," *SIGBED Rev.*, vol. 8, no. 1, pp. 54–61, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1967021.1967028

[4] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim, "Proartis: Probabilistically analyzable real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 94:1–94:26, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2465787.2465796

[5] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of real-time embedded systems," in *2012 IEEE 33rd Real-Time Systems Symposium*, Dec 2012, pp. 351–362.

[6] J. L. Diaz, D. F. Garcia, Kanghee Kim, Chang-Gun Lee, L. Lo Bello, J. M. Lopez, Sang Lyul Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, Dec 2002, pp. 289–300.

[7] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic schedulability analysis techniques for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 6, no. 1, pp. 1–53, 2019.