Poster Abstract: Run-time Dynamic WCET Estimation

Lia Cagnizi
lia.cagnizi@mail.polimi.it
Politecnico di Milano, Milano, Italy

Federico Reghenzani
federico.reghenzani@polimi.it
Politecnico di Milano, Milano, Italy
European Space Agency, Noordwijk, Netherlands

William Fornaciari
william.fornaciari@polimi.it
Politecnico di Milano, Milano, Italy

ABSTRACT

To guarantee the timing constraints of real-time IoT devices, engineers need to estimate the Worst-Case Execution Time. Such estimation is always very pessimistic and represents a condition that almost never occurs in practice. In this poster, we present a novel compiler-based approach that instruments the tasks to inform, at run-time, the operating system when non-worst-case branches are taken. The generated slack is then used to take better scheduling decisions.

CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; • **Software and its engineering** → **Real-time systems software**; **Compilers**; • **Hardware** → *Timing analysis.*

KEYWORDS

real-time scheduling, dynamic WCET, compiler transformation

ACM Reference Format:
Lia Cagnizi, Federico Reghenzani, and William Fornaciari. 2023. Poster Abstract: Run-time Dynamic WCET Estimation. In *International Conference on Internet-of-Things Design and Implementation (IoTDI '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3576842.3589168

1 INTRODUCTION

Many Internet-of-Things (IoT) devices must satisfy given timing constraints, which are usually expressed with the concept of a time *deadline*. Such IoT systems are then *real-time systems*. Mixed-Criticality Systems (MCS) are a particular class of real-time systems and have been a subject of research since the Vestal's article [4]. These systems integrate components with different criticality onto the same platform, where criticality is meant as the level of assurance that a software component must guarantee. To guarantee that timing constraints are met for real-time systems, the scheduling analysis must consider the Worst-Case Execution Time (WCET) of each task. This pessimistic assumption leads to a waste of system resources, for two reasons: tasks rarely execute their longest execution path, and the execution time analyses are usually very pessimistic, substantially over-estimating the real WCET [3]. In order to mitigate this problem, slack scheduling techniques have been developed to exploit unused WCET budgets [2]. MCS also targets this problem, with the aim to exploit over-provisioned resources and take advantage of the criticality concept. For example, the scheduler admits low criticality tasks during the slack time generated by high criticality tasks that do not run for their entire WCET [1]. However, the limitation of slack-reclaiming approaches is that the actual execution time of a task is known to the scheduler only after its completion. Instead, any info on the actual execution time before completion may help in performing a more efficient scheduling and allows more low criticality tasks to run.

In this article, we propose a compiler-based tool to automatically instrument software tasks in order to provide run-time WCET updates to the scheduler when non-worst-case paths are taken at run-time. Thanks to these new information, the scheduler could know, before the job completion, a more precise estimation of the WCET and produce a more efficient schedule which increases the total number of successfully completed tasks.

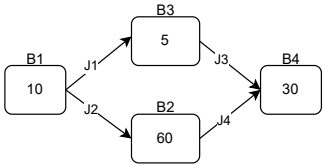


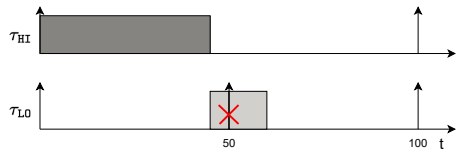
Figure 1: An example of CFG with four basic blocks (B1–B4) and four jump instructions (J1–J4). The number inside each block represent the worst-case execution time of that block.

2 METHODOLOGY

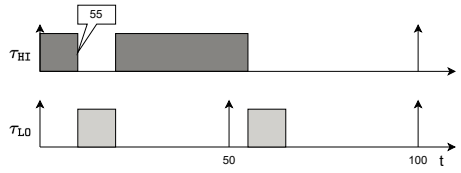
A system is composed of n tasks $\Gamma = \{\tau_1, \tau_2, ...\}$, each task is identified by the tuple (C_i, T_i, D_i, χ_i), where C_i is the WCET, T_i is the period or inter-arrival time, D_i is the relative deadline, χ_i the criticality level. In this paper, we consider an implicit deadline system, i.e., $D_i = T_i$. Regarding the criticality level, we consider a two-level model: $\chi_i \in \{\texttt{LO}, \texttt{HI}\}$. In the standard Vestal's model with two criticality levels, C_i is composed of two elements. In the context of this paper, we are not exploiting multiple values for the WCET, therefore, we consider C_i a scalar value corresponding to the highest criticality level WCET. Indeed, the criticality level, in this paper, only defines the importance of the task in the system and it is not used to provide different WCET estimations like the Vestal's model. A task generates a sequence of jobs, which represent the single units of computation. The k-th job of τ_i is denoted by τ_i^k and runs for $e_i^k < C_i$ time units. When a job completes before its WCET, it generates a slack time: $S_i^k = C_i - e_i^k$.

IoTDI '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0037-8/23/05…$15.00
https://doi.org/10.1145/3576842.3589168



(a) Without run-time WCET update



(b) With run-time WCET update

Figure 2: The illustration of one hyperperiod of Example 2.1.

2.1 A compiler-based run-time WCET update

We explain our approach by following the example depicted in Figure 1. The task WCET is computed by finding the worst-case execution path of the Control-Flow Graph (CFG), which is `B1-J2-B2-J4-B4` for a WCET of 100 time units. At run-time, whenever a taken branch of the CFG is not on the worst-case execution path, we already know that $S_i^k > 0$ at the future job completion. In our example, if at run-time the branch taken is J1, we can immediately calculate that e_i^k cannot be the WCET anymore but is upper-bounded by 45. Consequently, at the time that $J1$ is taken, we have the information that $S_i^k > 55$. If this information is immediately communicated to the scheduler, the scheduler can exploit it and potentially change the previous scheduling decisions.

Example 2.1. Let us consider the CFG of Figure 1 of a `HI`-crit task $\tau_{\texttt{HI}}$ having period and WCET $T_{\texttt{HI}} = C_{\texttt{HI}} = 100$. On the same system, a `LO`-crit task $\tau_{\texttt{LO}}$ with period $T_{\texttt{LO}} = 50$ and WCET $C_{\texttt{LO}} = 10$ is also present. Without the run-time information, $\tau_{\texttt{LO}}$ misses the deadline, because the scheduler has no information that the active job of $\tau_{\texttt{HI}}$ would finish before its WCET 100, and thus scheduling $\tau_{\texttt{LO}}$ before $\tau_{\texttt{HI}}$ would be speculative and non-safe. If the branch `J1` is taken, $e_{\texttt{HI}}^j = 45$: in traditional systems, this is only known at $t = 45$ thus $\tau_{\texttt{LO}}$ has no time to complete its execution. In our approach, $e_{\texttt{HI}}^1 = 45$ and $S_{\texttt{HI}}^1 = 55$ information is known at $t = 10$ thus $\tau_{\texttt{HI}}$ can be safely preempted and $\tau_{\texttt{LO}}$ scheduled immediately, allowing both tasks to meet the deadline. Figure 2 depicts the two situations.

We implemented this approach by developing a *pass* in the LLVM compiler. A *pass* is a compiler transformation modifying the so-called Intermediate Representation (IR). Modifying the IR has the advantage of being independent from the source programming language and the target machine architecture. The pass runs on the CFG of a task and injects, in each basic block that is not on the worst-case execution path, the calculation of the generated slack and a `call` instruction to a user-defined function of the operating system. The slack is passed to this function, which is in charge to inform the scheduler about the slack. More specifically, let B_i be a basic block and w_i its WCET. The pass runs for each B_i and calculates the cumulative WCET of each block: $\bar{w}_i = w_i + \max_{B_j : B_i \in \text{succ}(B_j)} \bar{w}_j$, where succ$(\cdot)$ are all the blocks successors of the argument. Note that the w_i of the first basic block of the CFG corresponds to the task WCET C_i. The pass creates a global variable p in the target code which keeps track, at run-time, of the remaining WCET at the end of the current block. Then, in each block B_i, which is not part of the worst-case execution path, the pass injects: (1) The calculation of the generated slack: $S_i = p - w_i$, which requires 2 IR instructions; (2) A `call` instruction to a function implemented in the OS, passing the S_i as parameter. To eliminate unnecessary updates of the variable p, the assignment instruction $p = \bar{w}_k - w_k$ is injected in a basic block B_k only if B_k has more than one successor. In summary, the total overhead is composed of 3 IR instructions for each modified basic block having only one successor, while in the case of multiple successors it can be up to 4.

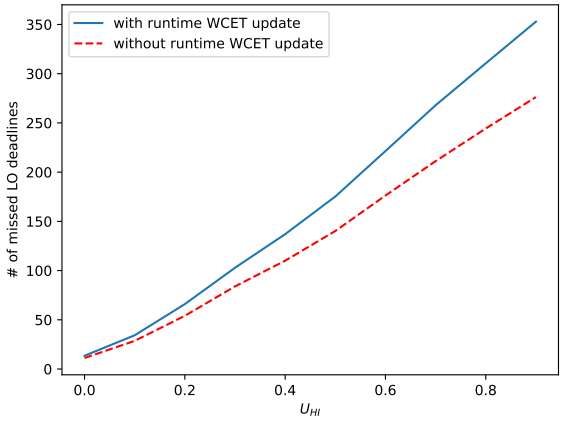


Figure 3: Results of the simulation.

2.2 Scheduling algorithm

We developed a single-core scheduling algorithm that is able to exploit the run-time WCET update. The scheduling policy is based on a hierarchical strategy: the tasks are split in two fixed-priority classes, `LO` and `HI`. Then, a traditional Earliest Deadline First (EDF) is applied in each class. The `HI`-crit tasks, having higher priority, always run and preempt `LO`-crit tasks. When no `LO`-crit workload is ready to run, `LO`-crit tasks are scheduled according to EDF. When the task is instrumented by the LLVM pass, the update function receiving the slack value S performs the following actions: (1) Check whether a task τ_x with $\chi_x = \texttt{LO}$ and $C_i < S$ exists; (2) If such a task exists, it is promoted to `HI` for one job execution; (3) The EDF scheduling policy is then reapplied by considering the new `HI` task.

3 EXPERIMENTAL RESULTS

We evaluated the performance of our approach via simulation. We simulated the behavior of an operating system that uses our proposed scheduling algorithm and implements the WCET updating strategy, comparing it to the normal scheduling. In particular, we implemented the standard EDF scheduler prioritized with criticalities and, then, our version with the WCET updater, evaluating the performance difference, in terms of scheduled jobs, between them.

The simulations are parameterized by the utilizations $U_{\texttt{HI}}$ and $U_{\texttt{LO}}$ defined in the usual way as $U_x = \sum_{\tau_i : \chi_i = x} \frac{C_i}{T_i}$. The number

of tasks is fixed at 5 LO-crit and 5 HI-crit. We perform a small experiment, not reported here for lack of space, that shows that the number of tasks has substantially no impact. We keep $U_{\texttt{LO}} = 1$ so that there is always a LO-crit task ready to run, while we varied $U_{\texttt{HI}}$ from 0 to 1. The averaged results of 20 000 simulations are depicted in Figure 3, that shows the number of jobs not able to complete their execution. The benefits of our approach are clearly visible as the utilization increases, and the number of jobs unable to complete the execution is reduced of 20% when $U_{\texttt{HI}}$ approaches 1.

4 CONCLUSIONS AND FUTURE WORKS

We addressed the problem of WCET over-estimation of real-time systems by providing a compiler-based tool that instruments the code to update at run-time the scheduler with the new WCET for the current job execution. We have tested our methodology on a simple scheduling algorithm obtaining a reduction up to 20% of the deadline misses. This tool opens several possible future research lines, including the study of other scheduling algorithms and the evaluation on a real board.

ACKNOWLEDGMENTS

This work has received funding from the EU High-Performance Computing Joint Undertaking (JU) under grant agreement No. 101033975 (EUPEX), European Space Agency (OSIP no. 4000133770 / 21 / NL / MH / hm), and ICSC National Research Center in High-Performance Computing.

REFERENCES

[1] Taeju Park and Soontae Kim. 2011. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *ACM International Conference on Embedded Software (EMSOFT)*. IEEE, Taipei, Taiwan, 253–262.

[2] Behnaz Ranjbar, Tuan D. A. Nguyen, Alireza Ejlali, and Akash Kumar. 2021. Power-Aware Runtime Scheduler for Mixed-Criticality Systems on Multicore Platform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 10 (2021), 2009–2023.

[3] Federico Reghenzani, Luca Santinelli, and William Fornaciari. 2020. Dealing with Uncertainty in PWCET Estimations. *ACM Trans. Embed. Comput. Syst.* 19, 5, Article 33 (sep 2020), 23 pages. https://doi.org/10.1145/3396234

[4] Steve Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE, Tucson, AZ, USA, 239–243.