Proceedings of the 7th Real-Time Scheduling Open Problems Seminar (RTSOPS)

held in conjunction with the 28th Euromicro Conference on Real-Time Systems (ECRTS)

Edited by Vincent Nelis[1] and Arvind Easwaran[2]
[1]*CISTER, ISEP, Portugal*
[2]*Nanyang Technological University, Singapore*

Toulouse, France
July 5, 2016

© Copyright 2016 CISTER, ISEP, Portugal, and Nanyang Technological University, Singapore.
All rights reserved. The copyright of this collection is with CISTER, ISEP and Nanyang Technological University. The copyright of the individual articles remains with their authors. Latex format for proceedings: Björn Brandenburg.

Message from the Chairs

It gives us great pleasure to introduce this volume that includes the Proceedings of the 7th Real-Time Scheduling Open Problems Seminar (RTSOPS 2016). This volume represents the continued openness in the Real-Time Systems research community to share and discuss unsolved problems concerning real-time scheduling theory.

This 7th edition of the seminar series is co-located with the 28th Euromicro Conference on Real-Time Systems (ECRTS 2016) that is being held in Toulouse, France. The day long seminar features 5 paper presentations and keynote presentations distributed over 2 sessions, a session on the impact of RTSOPS in real-time scheduling research and an open session for participants to present new scheduling problems and discuss status of known problems on-the-spot.

Each session includes ample collaboration time, during which we encourage all participants to interact in groups and tackle the presented problems. The hope is that we make headway in solving the problems and that more complete problem definitions and solutions will emerge as a result of the discussions initiated during the workshop.

We would like to thank the generosity of the Program Committee for their time and attention to detail that helped us assemble the program for the day. We are also grateful to the RTSOPS Steering Committee for their feedback and advice. This program would not have been possible without the efforts and support of the ECRTS 2016 organizing committee, in particular the General Chair Christian Fraboul.

We invite all of you to join us in taking advantage of this excellent opportunity to learn and interact with our fellow colleagues. We hope you enjoy RTSOPS 2016.

Vincent Nelis[1] and Arvind Easwaran[2]
[1]CISTER, ISEP, Portugal
[2]Nanyang Technological University, Singapore
RTSOPS 2016 Program Chairs

RTSOPS 2016 Technical Program Committee

Benny Akesson, Czech Technical University, Czech Republic

Björn Brandenburg, Max Planck Institute for Software Systems, Germany

Harini Ramaprasad, University of North Carolina at Charlotte, USA

Jinkyu Lee, Sungkyunkwan University, Republic of Korea

Patrick Meumeu Yomsi, CISTER/INESC TEC, ISEP, Portugal

Sebastian Altmeyer, University of Amsterdam, Netherlands

RTSOPS Steering Committee

Gerhard Fohler, TU Kaiserslautern, Germany

Liliana Cucu-Grosjean, INRIA Paris, France

Nathan Fisher, Wayne State University, USA

Robert Davis, University of York, UK

Technical Program

Efficient Execution of Dependent Tasks on Many-Core Processors
Hamza Rihani, Claire Maiza and Matthieu Moy 1

Tardiness Bounds for Global EDF Scheduling on a Uniform Multiprocessor
Kecheng Yang and James H. Anderson 3

The Feasibility Analysis of Mixed-Criticality Systems
Saravanan Ramanathan, Xiaozhe Gu and Arvind Easwaran 5

How Does Task Scheduling Affect Engine Control Performance?
Paolo Pazzaglia, Alessandro Biondi, Marco Di Natale and Giorgio Buttazzo 7

Johnson's Procedure: Mechanization and Parallelization
Mamoun Filali-Amine and Nabil Zaidi 9

Efficient Execution of Dependent Tasks on Many-Core Processors

Hamza Rihani, Claire Maiza, Matthieu Moy
Univ. Grenoble Alpes
CNRS, VERIMAG, F-38000 Grenoble, France
Email: {first}.{last}@imag.fr

I. Introduction

The increasing performance requirements of safety-critical real-time embedded systems made traditional single-core architectures obsolete. Moving to more complex many-core systems requires new techniques and tools for the certification of the embedded software. Timing and functional behaviours are subject to specific requirements of certification guidelines such as D-178B/C for avionics and ISO26262 for automotive systems. Determinism and predictability of such systems are a major challenge. Running tasks should be known in advance which makes the static and non-preemptive scheduling a suitable approach to reach an optimal execution with a guarantee of a certain degree of determinism. Recent work on mapping and scheduling problems [1], [2], [3] consider a known value (or a set of possible values) of the Worst-Case Response Time (WCRT) and computes a mapping that optimizes a predefined cost function. When the response time analysis is too pessimistic, the static scheduling may introduce an idle time which reduces the core utilization.

Scheduling techniques must rely on tight estimations of the WCRT which in turn depends on co-runner tasks. However, in order to obtain a tight upper-bound on the response time, a mapping and scheduling should be known in advance. Indeed, the response time is highly influenced by the co-runner tasks. Concurrent accesses to the same shared resource may introduce interferences that should be accounted for in the response time analysis. The search for an optimal scheduling with a tight WCRT analysis that includes the shared resource interferences is a challenging open problem.

II. The Open Problem

We illustrate the open problem on a Synchronous Data-Flow (SDF) application. SDF languages such as Lustre [4] offer an efficient programming paradigm that, using a certified compiler, can produce deterministic sequential code. An SDF application is represented with a task dependency graph, where the amount of exchanged data among the tasks is deterministic and known in advance. We consider the application to be running on a multi/many core architecture with a banked shared memory. Figure 1 represents an example of a node in an SDF application. The node increases the values of an input array by 1. Among the challenges while parallelizing such applications, we need to i) specify a task mapping that optimizes a certain cost function; ii) specify a scheduling per processing element that respects dependencies; iii) find a tight estimate of the response time that takes into account the interferences from co-runners. There exists several solutions for the mentioned points when taken individually. However, the connexion and the interaction among them remains as an open problem.

```
const n=1000;
node N (in_data: int^n) returns (out_data: int^n);
var N: int^n;
let
    N = 1^n; -- array of n values of 1
    out_data=in_data+N;
tel
```

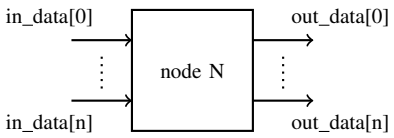

Fig. 1: Example of a Lustre node and its high level graphical representation

In this context, Figure 2 represents a potential solution for an efficient execution of such applications. The proposed technique relies on an iteration between a scheduling/mapping function and the WCRT analysis. Existing approaches may be used in each step but must interact altogether. In the following, we highlight open problems regarding such composability:

1) **Code Generation:** A code generator transforms the example given in Figure 1 into a low level language (such as C language) which in turn is compiled to run on a target architecture. We consider that `in_data` (produced by a previous node) is present in the local memory bank. After computation, `out_data` is copied to the next nodes' local memory banks. Several execution models can be applied: 1) A single phase execution model where the output is "sent" to the next node as soon as it is ready. 2) A structured execution model with an *execution phase* and a *replication phase*. The

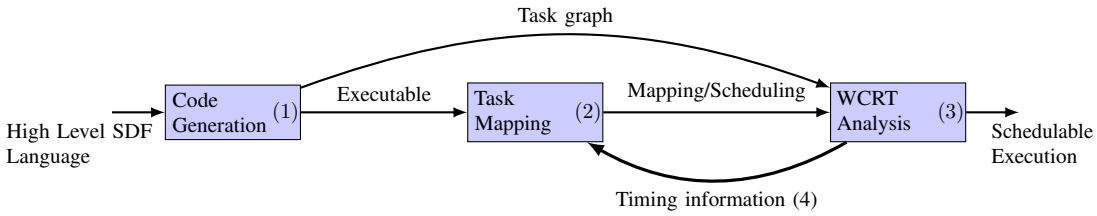

Fig. 2: High level work flow of a complete WCRT/Scheduling interaction

execution phase uses only the local memory bank to compute and store the output data. Then, a *replication phase* is added to copy the values of `out_data` into their memory destinations. The number of shared resource accesses may be significantly increased, but with a good scheduling of the execution and replication phases the interference may be reduced. The choice of the execution model affects the results in the next steps. It is not clear whether it is possible to recognize at this stage which execution model is more efficient by analyzing the structure of the application.

2) **Task Mapping:** We are interested in the optimization of the overall response time according to an estimated WCRT of each task. In memory intensive applications, the scheduler may consider minimizing the interference by avoiding concurrent accesses to the same shared resource. Each task is modeled by its release date, response time, deadline, and period. The single phase execution model may use traditional scheduling techniques to respect all the deadlines. The structured execution model offers more flexibility by co-scheduling communications and computations such that the interference is reduced [5]. However, we still need to investigate the efficiency of this method w.r.t a single-phase execution model where the number of accesses is not increased but the interference can be pessimistically overestimated.
3) **WCRT Analysis:** The WCRT analysis must be able to derive tight upper-bounds on the interference due to concurrent accesses to shared resources, and therefore estimate the response times and release dates for all tasks. This requires a task mapping and scheduling as well as the constraints on the task dependencies. The execution model given by the code generation step can highly influence the analysis. In the case of a structured execution model, the upper bounds on the shared resource interferences in the execution and replication phases may be too pessimistic due to pessimistic assumptions on the arbitration policy. This may as well result in a larger estimation on the overall interference per task depending on the application's topology. As a result, one may run the analysis on both execution models and retain the one with the best overall response time.
4) **WCRT/Mapping Iteration:** Depending on the chosen task mapping and scheduling per processing elements, the response times must be re-estimated by taking into account the interference in order to check that all constraints are respected and potentially reduce idle times. However, the optimality obtained from step (2) may be lost due to the updated timing information. A possible solution relies on constraint programming languages. A solver finds a satisfiable task mapping that optimizes the cost function. The WCRT Analysis and a schedulability test can then compute a tighter estimation by taking into account potential interferences. New timing constraints are added to the model and the search stops when no better solution can be found.

Since the mapping problem is NP-hard, it is hard to prove that the iteration done in $4)$ converges toward an absolute optimal solution (assuming its existence). In the absence of such guarantee, one can iterate for a certain amount of time and choose the best solution. This interaction remains as an open problem.

Acknowledgment

This work was funded by the grant CAPACITES (PIA-FSN2 n°P3425-146798) from the French *Ministère de l'économie, des finances et de l'industrie*.

References

[1] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, Sept 2013, pp. 1–15.
[2] W. Puffitsch, E. Noulard, and C. Pagetti, "Mapping a multi-rate synchronous language to a many-core processor," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pp. 293–302.
[3] J. Walter and W. Nebel, "Energy–aware mapping and scheduling of large–scale macro data–flow applications," in *1st International Workshop on Investigating Dataflow in Embedded Computing Architecture*, 2015.
[4] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language lustre," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, Sep 1991.
[5] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, "Memory-processor co-scheduling in fixed priority systems," in *23rd ACM International Conference on Real-Time Networks and Systems (RTNS)*, Lille, France, November, 2015.

Tardiness Bounds for Global EDF Scheduling on a Uniform Multiprocessor

Kecheng Yang and James H. Anderson
Department of Computer Science
University of North Carolina at Chapel Hill, USA
Email: {yangk, anderson}@cs.unc.edu

I. Introduction

Since the multicore revolution, the focus of research on real-time scheduling has shifted to multiprocessor platforms. *Global earliest-deadline-first (GEDF)* is a widely studied multiprocessor scheduling algorithm. Unfortunately, in hard real-time (HRT) systems where every deadline must be met, platform utilization must be restricted in order to guarantee schedulability under GEDF on a multiprocessor platform due to the Dhall Effect [2]. Nonetheless, in soft real-time (SRT) systems, where deadlines may be missed provided deadline tardiness is provably bounded, Devi and Anderson [1] showed the SRT-optimality of GEDF schedulers, *i.e.*, deadline tardiness bounds are guaranteed under GEDF scheduling for any feasible system.

The SRT-optimality result applies to both preemptive and non-preemptive GEDF schedulers, but only on a multiprocessor consisting of *identical* processors. The *uniform* multiprocessor model, which extends the identical multiprocessor model by associating a *speed* with each processor, is a widely studied heterogeneous multiprocessor model. We are interested in whether GEDF schedulers are still SRT-optimal on a uniform multiprocessor.

II. System Model

On a uniform multiprocessor platform, processors may have different speeds. The *speed* of processor p is denoted as s_p. Identical multiprocessors are a special case of uniform platforms, where all processors happen to have the same speed, which is usually normalized as 1.0. We consider the scheduling of n sequential sporadic tasks on a uniform multiprocessor platform of m processors. Each task τ_i is specified by (C_i, T_i), where C_i is its *worst-case execution requirement*, which is defined as its worst-case execution time on a unit-speed processor, and T_i is its *period*, or minimum separation between any two consecutive invocations of the task. Periodic tasks are a special case of sporadic tasks where the separation between any two consecutive invocations of a task τ_i is always *exact* T_i. We assume the deadlines to be *implicit* (*i.e.*, the relative deadline of each task equals its period).

A job is an invocation of a task. The jth job of task τ_i is denoted as $\tau_{i,j}$ and is released at time $r_{i,j}$ with an absolute deadline $d_{i,j} = r_{i,j} + T_i$. If $\tau_{i,j}$ completes at time t_c, then its *tardiness* is defined as $\max\{0, t_c - d_{i,j}\}$. The tardiness of a task is the maximum tardiness of any of its jobs. A job is *ready* if and only if it is released and all previous jobs of the same task have already completed. Only ready jobs can commence execution since we assume each individual task to be sequential.

The utilization of a task τ_i is defined as $u_i = C_i/T_i$. Note that, on a uniform platform, $u_i \leq 1$ is not necessarily required for a system to be feasible. A feasibility condition, which depends on processor speeds, is provided by the following theorem cited from [3].

Theorem 1 (Theorem 4 in [3])**.** *Let S_k denote the sum of the k largest processor speeds, and let U_k denote the sum of the k largest task utilizations. Then, the n tasks can be scheduled to meet all deadlines on the m uniform processors, if and only if the following constraints hold:*

$$U_n \leq S_m, \tag{1}$$

$$U_k \leq S_k, \text{ for } k = 1, 2, \ldots, m-1. \tag{2}$$

The above feasibility condition was established for *periodic* tasks in *HRT* systems. Nonetheless, it is easy to show that (1) and (2) are also a necessary and sufficient feasibility condition for *sporadic* tasks in *SRT* systems.

Migrations. On an identical multiprocessor, a scheduler needs to determine *which* jobs are scheduled, but *where* they are scheduled matters less since every processor is the same. In contrast, on a uniform multiprocessor, *where* jobs are scheduled is crucial, and must be explicitly specified by a scheduler. In this context, we define that, under *non-preemptive* scheduling, once a job is scheduled on a particular processor, it continuously executes on that processor until complete.

III. Known Results and The Open Problem

Although both preemptive and non-preemptive GEDF schedulers are SRT-optimal on *identical* multiprocessors [1], it is shown in [6] that, on *uniform* platforms, there exists a feasible system where a task can become unboundedly tardy under

non-preemptive GEDF scheduling, which precludes the SRT-optimality of the non-preemptive GEDF scheduler. Therefore, we shift our focus to a fully preemptive GEDF scheduling algorithm **(A)** as follows.

(A) If at most m jobs are ready, then all ready jobs are scheduled; otherwise, the m ready jobs with earliest deadlines are scheduled. Migrations are allowed, by which the ready job with the kth earliest deadline is always scheduled on the kth fastest processor for any k. Deadline ties are broken arbitrarily.

Thus, we state the following open problem:

> *Under the preemptive GEDF scheduling algorithm* **(A)***, is the deadline tardiness for every task provably bounded for any feasible system on a uniform multiprocessor?*

Some variants of this open problem have been solved in prior work, such as,

- relaxing the task model to allow intra-task parallelism [5];
- constraining per-task utilizations [4];
- restricting the platform to have only two uniform processors [6].

Therefore, to clarify, we would like to emphasize the following assumptions in this open problem:

- intra-task parallelism is strictly forbidden (jobs of the same task must execute in sequence);
- no constraints on task utilizations other than the feasibility condition (*i.e.*, (1) and (2)) are assumed;
- the uniform platform may have more than two processors.

To solve this open problem, we need to consider both potential answers. In order to answer "No" to this open problem, we need a counterexample where at least one task in a feasible system becomes unboundedly tardy under the preemptive GEDF scheduling algorithm **(A)**. The potential counterexample must have at least three uniform processors and likely has multiple high-utilization individual tasks (utilizations higher than the speeds of some processors in the system). On the other hand, in order to answer "Yes" to this open problem, we need to handle a key difference between an identical multiprocessor and a uniform one, as described in next section.

IV. The Key Difference

The difficulty in extending the SRT-optimality result on identical multiprocessors [1] to uniform multiprocessors comes from the change of feasibility condition. In the feasibility condition for scheduling a sporadic implicit-deadline task set on m identical processors, the *total utilization constraint*, which corresponds to (1), is that the sum of the utilizations of all tasks is at most m; the *per-task utilization constraint*, which corresponds to (2), is that the utilization of each individual task is at most 1.0. These two utilization constraints underlie the proof framework in [1].

In the feasibility condition for uniform platforms, (1) can play a similar role as the total utilization constraint in the identical case; however, (2) is not as convenient as the per-task utilization constraint in the identical case. Specifically, the per-task utilization constraint for an identical multiprocessor guarantees the following property.

(P) If any job $\tau_{i,j}$ continuously executes, it must complete within T_i time units, regardless of the processor on which $\tau_{i,j}$ executes.

This property is critical in the induction-based proof framework in [1], in order to handle the precedence constraint of each sequential sporadic task (*i.e.*, a job cannot commence execution until all previous jobs of the same task have already completed). However, **(P)** does not necessarily hold for a feasible system on a uniform multiprocessor.

Example 1. *We consider the scheduling of three sporadic tasks with utilizations $u_1 = u_2 = u_3 = 2$ on three uniform processors with speeds $s_1 = 4$ and $s_2 = s_3 = 1$. By checking* (1) *and* (2)*, this system is feasible. However,* **(P)** *does not hold when any job entirely executes on processor* 2 *or* 3*.*

Acknowledgment

This work is supported by NSF grants CPS 1239135, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and funding from General Motors.

References

[1] U. Devi and J. Anderson, "Tardiness bounds for global EDF scheduling on a multiprocessor," *Proceedings of the 26th IEEE Real-Time Systems Symposium*, 2005.

[2] S. Dhall and C. Liu, "On a real-time scheduling problem," *Operations Research*, 1978.

[3] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001.

[4] G. Tong and C. Liu, "Supporting soft real-time sporadic task systems on heterogeneous multiprocessors with no uilitzation loss," *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[5] K. Yang and J. Anderson, "Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors," *Proceedings of the 12th IEEE Symposium on Embedded Systems for Real-Time Multimedia*, 2014.

[6] K. Yang and J. Anderson, "On the soft real-time optimality of global EDF on multiprocessors: From identical to uniform heterogeneous," *Proceedings of the 21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015.

The Feasibility Analysis of Mixed-Criticality Systems

Saravanan Ramanathan, Xiaozhe Gu, Arvind Easwaran
Nanyang Technological University, Singapore
Email: saravana016@e.ntu.edu.sg, guxi0002@e.ntu.edu.sg, arvinde@ntu.edu.sg

I. Introduction

Mixed-Criticality (MC) scheduling has received a lot of attention since Vestal introduced the MC task model [1]. There have been several studies focusing on the scheduling problem for the MC task model [2]. Baruah et al. [3] have showed that the scheduling problem for MC systems is NP-Hard. Ekberg et al. [4] presented a note on open problems in MC systems. Till date there is neither an optimal scheduling policy nor a discussion on the feasibility analysis for MC task systems. In this work, we address this problem by presenting a possible solution to design a non-clairvoyant[1] optimal algorithm for scheduling implicit-deadline periodic MC task systems on uniprocessor platform and derive the feasibility tests to determine whether a given task system is MC schedulable.

Task Model: We consider the problem of scheduling dual-criticality (namely LO and HI) periodic task set τ on a uniprocessor platform. Each MC task τ_i represents an infinite number of job releases and is characterized by a tuple $(T_i, \chi_i, C_i^L, C_i^H, D_i)$, where T_i is the period, $\chi_i \in \{LO, HI\}$ is the criticality-level, C_i^L is the LO-criticality Worst-Case Execution Time (WCET), C_i^H is the HI-criticality WCET; we assume $C_i^L \leq C_i^H$ and $D_i (= T_i)$ is the deadline of the task. LO- and HI-criticality utilization of a task τ_i is defined as $u_i^L \stackrel{\text{def}}{=} C_i^L/T_i$ and $u_i^H \stackrel{\text{def}}{=} C_i^H/T_i$ respectively. System-level utilizations are defined as $U_L^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L$, $U_H^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L$ and $U_H^H \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H$.

II. Problem Description

Known results: The only known necessary feasibility condition for scheduling MC task systems is given by $U_L^L + U_H^L \leq 1$ and $U_H^H \leq 1$. It has been shown that algorithms such as EDF_VD [5] for uniprocessor and MC-Fluid [6] and MCF [7] for multiprocessor MC systems are speed-up optimal[2] with a speed-up bound[3] of $4/3$. That is, these algorithms can successfully schedule any MC task system that satisfy the conditions $U_L^L + U_H^L \leq 3/4$ and $U_H^H \leq 3/4$. Figure 1 shows the feasibility region of MC task systems. The diagonally shaded region in the figure is our region of interest to determine the feasibility of a task system.

Open problem: What is a tight feasibility bound for MC task systems? The existing MC scheduling algorithms are only speed-up optimal, but not optimal in general, i.e. there are feasible task sets that are not schedulable by these algorithms. Therefore, to determine the feasible region of MC task systems we need a non-clairvoyant optimal scheduling algorithm with an exact schedulability test. Given the complexity of MC scheduling problem, it is reasonable to first consider the periodic task systems which has a fixed job release sequence. As a first step, it is necessary to design a non-clairvoyant optimal algorithm for scheduling implicit-deadline periodic task systems on uniprocessor platform and later extend it to the multiprocessor platform and sporadic task systems.

Challenge: The major challenge in deriving the feasibility test for MC system is in determining the worst-case pattern. So far, we do not know of any such worst-case scenario either for periodic or for sporadic MC task systems. Since the mode switch instant is not known a priori, we need to focus on determining the worst-case scenario for the non-clairvoyant behavior.

III. Possible Solutions

This section briefly describes the possible solution in designing a non-clairvoyant optimal scheduling algorithm for periodic MC task systems. We propose a fluid-model based scheduling algorithm.

The dual-rate fluid model [6] was designed to schedule implicit-deadline task systems with two criticality levels. It assigns different execution rates to tasks in each criticality level. Under fluid scheduling the execution rates are computed by determining the worst-case mode switch instant of HI-criticality tasks. Figure 2 shows the worst-case mode switch instant of a HI-criticality task, where θ_i^L and θ_i^H represents the execution rates of HI-criticality task in LO and HI mode respectively. The intuition behind this is that the carry-over job executes with the HI mode rate for the shortest possible duration.

[1] Non-clairvoyant algorithm has no knowledge on when the mode switch will be triggered.
[2] If an algorithm has an optimal speed-up bound, then no non-clairvoyant algorithm can have a tighter speed-up bound.
[3] Speed-up bound is defined as the additional processor speed required to schedule a feasible taskset using the proposed algorithm [8].

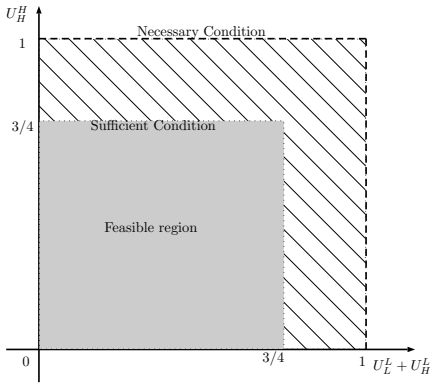

Fig. 1. Feasibility region for MC system

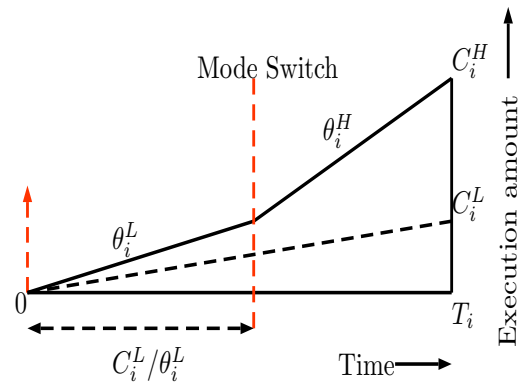

Fig. 2. Worst-case mode switch instant of a HI-criticality task

Under fluid scheduling, the execution rates determine the mode switch instant. That is, each HI-criticality task complete its C_i^L units of execution at C_i^L/θ_i^L. At that instant, if the task does not signal completion the mode switch will be triggered. We refer to this time instant as the latest possible virtual deadline of a HI-criticality task in LO mode.

The design of optimal scheduling algorithm involves three conditions:

- In LO mode: Schedule LO-criticality tasks as late as possible.
- In LO mode: Schedule HI-criticality tasks with their virtual deadline (C_i^L/θ_i^L).
- In HI mode: Optimal scheduling of HI-criticality tasks inclusive of carry-over demand of HI-criticality tasks.

In the case of non-MC systems, fluid scheduling is known to be optimal for multiprocessor platform [9],[10]. For MC systems, it is possible to optimally schedule HI-criticality tasks in HI mode using fluid scheduling by considering the worst-case mode switch instant of all HI-criticality tasks.

In order for each HI-criticality task to meet its deadline in HI mode, the restriction imposed on fluid scheduling is that each HI-criticality task must execute its C_i^L budget within C_i^L/θ_i^L time units from its release. Failure to allocate C_i^L units within this deadline will result in a deadline miss in HI mode.

In non-MC systems, algorithms such as DP-Fair [9] and Bfair [10] are used to convert fluid schedule into non-fluid schedule without any loss in schedulability. The intuition behind this is that between two boundaries, tasks are allocated execution units equal to the product of its execution rate and length of the duration. The tasks are guaranteed proportional execution units across all job deadlines. For MC systems, in addition to job deadlines, HI-criticality tasks need to be guaranteed proportional execution units across all virtual deadlines (C_i^L/θ_i^L). Since we consider periodic job releases, one possible way to guarantee fairness across boundaries would be to consider each HI-criticality job as a set of sub jobs with same release time and different execution requirements and deadlines.

By guaranteeing fairness across both job and virtual deadline of each HI task and executing the LO-criticality tasks as late as possible in LO mode using any work conserving scheduling algorithm, it is possible to optimally schedule the task system. We shall then derive the necessary and sufficient feasibility tests for implicit-deadline periodic MC task systems. For sporadic MC task systems the derived feasibility tests becomes a necessary (but not sufficient) condition. We believe that this is a potential research direction for MC scheduling.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium (RTSS), 28th IEEE International*, Dec 2007.

[2] A. Burns and R. I. Davis. (2013) Mixed Criticality Systems - A Review. http://www-users.cs.york.ac.uk/burns/review.pdf.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *Computers, IEEE Transactions on*, vol. 61, no. 8, Aug 2012.

[4] P. Ekberg and W. Yi, "A note on some open problems in mixed-criticality scheduling," in *Real-Time Scheduling Open Problems Seminar (RTSOPS), 6th International*, July 2015.

[5] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Real-Time Systems (ECRTS), 24th Euromicro Conference on*, July 2012.

[6] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee, "MC-Fluid: Fluid Model-Based Mixed-Criticality Scheduling on Multiprocessors," in *Real-Time Systems Symposium (RTSS), 35th IEEE International*, Dec 2014.

[7] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *Real-Time Systems Symposium (RTSS), 36th IEEE International*, Dec 2015.

[8] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance [scheduling problems]," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, Oct 1995.

[9] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: a unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Systems*, vol. 47, no. 5, 2011.

[10] G. Nelissen, H. Su, Y. Guo, D. Zhu, V. Nélis, and J. Goossens, "An optimal boundary fair scheduling," *Real-Time Systems*, vol. 50, no. 4, 2014.

How Does Task Scheduling Affect Engine Control Performance?

Paolo Pazzaglia, Alessandro Biondi, Marco Di Natale and Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {name.surname}@sssup.it

***Abstract*—The timing behavior of engine control applications has been analyzed in several papers, whose results allow verifying the system schedulability under different schedulers and assumptions. However, the characterization of the scheduling problem suffers from several inaccuracies, the most important one being probably the hard deadline assumption. In fact, in engine control, a few deadline misses are often well tolerated and the objective of the scheduling is not necessarily to meet all deadlines, but rather maximizing the engine performance. This requires evaluating the impact of the scheduler, which is much more complex, since it does not refer to timing properties in isolation, but is related to the functional behavior and overload management. The open problem addressed here is to understand how scheduling decisions affect the engine performance. A possible approach to address such a problem is through a suitable tool chain that integrates a scheduling simulation framework with an engine model for estimating the impact of a scheduling policy on engine performance.**

I. Introduction

Engine control systems require actions that need to be performed at specific angular rotations of the crankshaft, in addition to other regular periodic activities. As a consequence, a typical engine control application consists of time-driven periodic tasks with fixed periods, typically between a few milliseconds and 100 ms (see [1], page 152), and angular tasks triggered at specific crankshaft angles. The activation rate of such angular tasks hence varies with the engine speed (variable-rate tasks). For example, for engines where the speed varies from 500 to 6500 revolutions per minute (RPM), the interarrival times of the angular tasks range from about 10 to 120 ms (assuming a single activation per cycle).

To prevent overload conditions at high engine speed, such angular tasks are implemented in such a way to decrease their computational requirements for increasing speeds [2], hence they are referred to as adaptive variable-rate (AVR) tasks. In particular, they are implemented as a set of operational modes, each characterized by a set of functions operating within a given speed range.

Analyzing the schedulability of tasks sets consisting of both periodic and AVR tasks is a difficult problem that has been addressed by several authors under various simplifying assumptions, under both fixed priority scheduling [3]–[5] and Earliest Deadline First (EDF) [6]–[8].

All the papers considered above, however, focused on analyzing the schedulability of task sets consisting of periodic and AVR tasks, without any concern on engine performance.

A performance-driven design approach has been addressed in [9] for finding the transition speeds that trigger the mode changes of an AVR task.

The challenging open problem addressed here is to understand how scheduling decisions may affect engine performance, in terms of power, fuel consumption, and pollution. Solving this problem requires the integration of different crucial components, including a realistic engine model, schedulability analysis, control algorithms, and a simulation environment that accounts for the delays introduced by the operating system.

II. Open problems

The analysis of engine control applications is an interesting and challenging problem. Not only the scheduling problem is relatively new and complex (due to the presence of AVR tasks), but the application is characterized by multiple performance criteria (power, emissions, fuel consumption, noise) and deadline misses are clearly tolerated by the control logic. In addition, when studying the impact of time delays on the performance, details on the functional and code implementation of the controls cannot be avoided. Depending on them, a deadline miss can result in skipping an actuation, or performing an actuation with old data, or even with incomplete and approximate information available.

The impact of all these issues on the performance criteria of the engine is far from obvious. Likely, most of the computational models and their corresponding real-time analyses that are available today, fail in accounting for all the parameters that affect the engine performance.

Similarly, the performance indices that are more affected by scheduling issues are to date, and to the best of our knowledge, unknown.

Finally, further complications arise when considering the adaptive behavior of engine control tasks. As discussed in Section I, to avoid overload conditions, AVR tasks perform a mode-change among a set of possible control implementations. As the engine speed increases, AVR tasks switch to simplified (or even different) control implementations with a reduced computational demand. The switch happens at given *transition speeds*. This determines a trade-off between system schedulability (or in general, response-times and deadline miss ratio) and the resulting performance of the engine. The design problem of transition speeds has been investigated in [9] under hard real-time schedulability constraints. Alternative

approaches based on extended analyses to cope with temporary overaload conditions are still not explored.

III. Proposed approach and outstanding issues

To solve the problem of how to evaluate the impact of deadline miss and scheduling delays on the performance of the CPS system there are several possible approaches. One possibility is to attempt a functional characterization of the dependency of the performance function(s) from the temporal parameters. For some simple control systems this is possible, but for realistic CPS the task soon becomes prohibitively difficult because of several reasons, including the need to satisfy multiple performance criteria and the difficulty in finding analytical models for the impact of late task executions on other tasks (this is why several papers of this type restrict the analysis to scheduling with time isolation).

Fuel injection is a prototypical case in this sense for the reasons discussed in the previous section.

We are attempting a first analysis of these issues by building a sample experimental framework. The envisioned approach for the evaluation of the impact of scheduling policies is based on a cosimulation framework that, following the principles of CPS system analysis, includes a set of models:

- A model of the engine and the combustion process in it (the physical system or plant)
- A model of the engine controls
- A model of the task configuration and the scheduling

The system is based on the Simulink toolset and leverages the T-Res environment for the simulation of the task scheduling. For the development of the engine model we leveraged information from several sources, including engine models for the steady state and event-based models as described in [1] and other empirical models found online. The engine controls are currently extremely simple and only contains a simple analytical formula that computes the angle of injection and the injection time that is defined by a calibration table.

The T-Res simulation framework is described in [10]. For the purpose of this project we extended the task model block and the timing information associated with the task to allow for the modeling of the AVR behavior. The task block in T-Res includes a signal for the explicit activation in case of event-triggered tasks, and this signal is used to define the activation of the task in correspondence to defined position of the engine crankshaft. In addition, the block has been extended to include another imput that refers to a generic *mode*. This input can be used for multiple purposes and defines in general a different execution time behavior for a finite and enumerated set of conditions. In our case, the mode index is provided from a simple block that looks at the engine rotation speeds and, based on the speed range, defines the execution time that the task requires.

Currently, within the assumptions of our model, the simulation is able to show how the scheduling delays result in errors in the angle/duration of the injection actuation (as shown in Figure 1, which aims at showing that adaptation of the task WCET for higher rates is indeed required to reduce the errors). However, we are still not able to express how a more complex implementation (WCET) also translates into a possible more accurate control law.

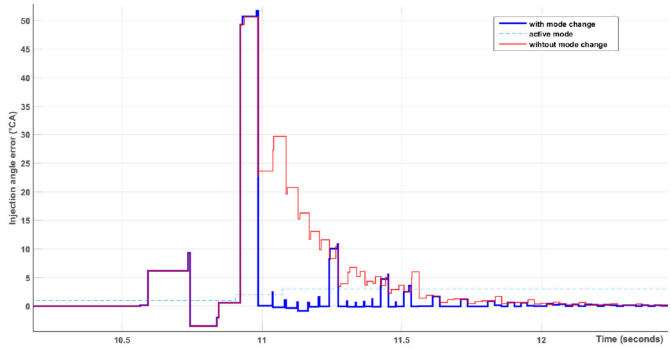

Figure 1. Angle error with scheduling delays with and without adaptation in the task execution times.

Future directions: While the framework and the approach are quite general, it is not clear what amount of work is reusable and if the problem can be partitioned in such a way that the timing problem can be separated from the functional analysis. In this case, the simulations with the model of the system and the controls could be used to derive a formulation of the dependency of the performance(s) with respect to the timing parameters (and select the relevant parameters). The performance function(s) could then be used to evaluate the scheduling solutions.

References

[1] L. Guzzella and C. H. Onder, *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer-Verlag, 2010.

[2] D. Buttle, "Real-time in the prime-time," in *Keynote speech at the 24th Euromicro Conference on Real-Time Systems*, Pisa, Italy, July 12, 2012.

[3] J. Kim, K. Lakshmanan, and R. Rajkumar, "Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems," in *Proc. of the Third IEEE/ACM Int. Conference on Cyber-Physical Systems (ICCPS 2012)*, Beijing, China, April 2012, pp. 28–38.

[4] R. I. Davis, T. Feld, V. Pollex, and F. Slomka, "Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling," in *Proc. 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, Berlin, Germany, April 2014.

[5] A. Biondi, M. D. Natale, and G. Buttazzo, "Response-time analysis for real-time tasks in engine control applications," in *Proceedings of the 6th International Conference on Cyber-Physical Systems (ICCPS 2015)*, Seattle, Washington, USA, April 14-16, 2015.

[6] G. Buttazzo, E. Bini, and D. Buttle, "Rate-adaptive tasks: Model, analysis, and design issues," in *Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE 2014)*, Dresden, Germany, March 24-28, 2014.

[7] A. Biondi and G. Buttazzo, "Engine control: Task modeling and analysis," in *Proc. of the International Conference on Design, Automation and Test in Europe (DATE 2015)*, Grenoble, France, March 9-13, 2015, pp. 525–530.

[8] A. Biondi, G. Buttazzo, and S. Simoncelli, "Feasibility analysis of engine control tasks under EDF scheduling," in *Proc. of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10, 2015.

[9] A. Biondi, M. D. Natale, and G. Buttazzo, "Performance-driven design of engine control tasks," in *Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS 2016)*, Vienna, Austria, April 11-14, 2016.

[10] F. Cremona, M. Morelli, and M. D. Natale, "Tres: A modular representation of schedulers, tasks, and messages to control simulations in simulink," in *Proc. of the 31st ACM Symposium on Applied Computing (SAC 2016)*, Pisa, Italy, April 4-8, 2016.

Johnson's procedure: mechanization and parallelization

Mamoun Filali Amine
IRIT
Université de Toulouse, France
Email: filali@irit.fr

Nabil Zaidi
Lycée Blaise Pascal
Clermont-Ferrand, France
Email: nabilzaidi@live.fr

Abstract

In this note, we present a mechanization of Johnson's procedure. We first specify the scheduling problem, we are concerned with. Then, we formalize Johnson's procedure and prove an invariant. Thanks to this invariant, we prove the correctness of Johnson's procedure and its optimality. Last, also thanks to the invariant, we elaborate a concurrent version of Johnson's procedure which is correct by construction. We use the Python language for expression and experimentation purposes. The Isabelle HOL assistant theorem prover is used for validating the correctness.

I. Introduction

Johnson's procedure [1] is one of the oldest procedure that has been proposed for scheduling in an optimal way a list of jobs over two machines. Although Johnson's procedure is described very precisely in the seminal paper [1] and in many scheduling textbooks [2], to the best of our knowledge, the assertional reasoning establishing its correctness with respect to a given program text has not been published. Moreover, we are aware neither of a mechanization[1] of such a proof nor of a parallelization of such a procedure.

This kind of scheduling is used inside multicores for the instructions to be processed. The calculus of the worst case execution time (WCET) for such processors relies on techniques: ILP, constraints resolution, ... for dealing with such a complex problem. Recently, the analysis of the scheduling of trains over rails has also been considered through similar techniques. Also, one technique to address the precise analysis of real time problems is that of timed automata. In order to scale such a technique, it seems promising to reuse the known optimality results for such a problem. Last, if we consider the certification of critical real-time systems, the use of formal methods is by now mandatory (DO178C). The mechanization of the basic procedures of the domain and their parallelization can make easier such certifications. From our point of view, the certification of real-time algorithms and their parallelization is an interesting open problem to address.

II. Specification

Notations

- A list l is a permutation of l' is denoted: $l \sim l'$ (l and l' are also said to be similar). Given two lists $l1$, $l2$, their concatenation is denoted $l1 + l2$. Given an element e, the list consisting of the unique element e ist denoted [e].
- Given a couple c of elements, the first (resp. second) component of c is denoted c_1(resp. c_2).
- Given a list l of couples, l is non decreasing(resp. non increasing) according to the first element(resp. to the second element) is denoted $\mathbf{inc}_1(l) = \mathbf{inc}(\mathbf{map\ fst}\ l))$ (resp. $\mathbf{dec}_2(l) = \mathbf{dec}(\mathbf{map\ snd}\ l)$).

A. Expression.

In the following, we model a job by a couple of naturals. The first (resp. the second) element of the couple is the duration required on the first (resp. the second) machine. The jobs to be scheduled are modeled as a list of natural couples.

1) Makespan definition: The makespan function **ms** is defined through an intermediate definition $\mathbf{ms}_i$ as follows:

$$\begin{aligned}\mathbf{ms}_i \in\ & (\mathbb{N}\times\mathbb{N})\text{list}\times(\mathbb{N}\times\mathbb{N})\to\mathbb{N}\times\mathbb{N}\\ & (l,i)\mapsto \mathbf{reduce}(\mathbf{lambda}(m,m'):\mathbf{lambda}(d,d'):(m+d,\max(m',m+d)+d')),l,i)\end{aligned}$$

and $\mathbf{ms}(l) = \mathbf{ms}_i(l,(0,0))$.

2) Scheduling specification: We model Johnson's procedure as a function **J** with the following properties:

(J1) **J** is a total function: $\mathbf{J} \in (\mathbb{N}\times\mathbb{N})\text{list} \to (\mathbb{N}\times\mathbb{N})\text{list}$.

(J2) **J** is conservative: **J** does not create or loose jobs. **J** generates a permutation: $\forall l.\ \mathbf{J}(l) \simeq l$.

(J3) **J** is optimal: $\forall l\ l'.\ l \sim l' \Rightarrow \mathbf{ms}(\mathbf{J}(l))_1 \leq \mathbf{ms}(l')_1 \wedge \mathbf{ms}(\mathbf{J}(l))_2 \leq \mathbf{ms}(l')_2$.

[1]Actually, we have done a mechanization of all the development in Isabelle HOL. Due to the lack of space, we do not detail it here.

III. Johnson's procedure expression and correctness

In order to establish the correctness of Johnson's procedure, we first express it is as a recursive Python procedure. Then, we state an invariant characterizing a schedule as returned by Johnson's procedure.

A. Johnson's procedure functional expression

```
# auxiliary procedure
def J_aux(b,l,e): # b is to be scheduled first, l remains to process, e is to be scheduled last
    # recursive procedure : the variant is len(l)
    if l == []: return b+e
    else:
        m1,m2 = min (l, key = lambda (d1,d2) : d1 ), min (l, key = lambda (d1,d2) : d2 )
        if m1[0] <= m2[1]: l.remove(m1); return J_aux(b+[m1],l,e)
        else: l.remove(m2); return J_aux(b,l,[m2]+e)
# Johnson's procedure
def J(l): return J_aux([],l,[])
```

B. Johnson's procedure invariant

The invariant (I_J) states formally how the schedule returned by Johnson's procedure is structured. This schedule is indeed a 2-partition of the jobs characterized by the comparison ($\leq$) of the first duration and the second duration of a job. The first subset is sorted according to the increasing order of the first duration, while the second subset is sorted according to the decreasing order of the second duration. More formally:

$$
(I_J) \quad \begin{array}{ll} & \mathbf{J}(l) \sim l \\ \wedge & \mathbf{let}\ l1 = [(d1, d2)\ \mathbf{for}\ (d1, d2)\ \mathbf{in}\ \mathbf{J}(l)\ \mathbf{if}\ d1 \leq d2]\ \mathbf{in} \\ & \mathbf{let}\ l2 = [(d1, d2)\ \mathbf{for}\ (d1, d2)\ \mathbf{in}\ \mathbf{J}(l)\ \mathbf{if}\ \neg(d1 \leq d2)]\ \mathbf{in} \\ & \quad \mathbf{J}(l) = l1@l2 \\ \wedge & \mathbf{inc}_1(l1) \wedge \mathbf{dec}_2(l2) \end{array}
$$

C. Johnson's procedure proof obligations

- The properties (J1) and (J2) are trivially established thanks to the invariant of the auxiliary procedure `J_aux` and the invariant (I_J) seen in the previous section.
- The proof of property (J3) relies on the following intermediate result stating that the Johnson's procedure is efficient:

  $$\forall l.\ \mathbf{ms}(\mathbf{J}(l))_1 \leq \mathbf{ms}(l)_1 \wedge \mathbf{ms}(\mathbf{J}(l))_2 \leq \mathbf{ms}(l)_2$$

  In turn, this result relies on the following property (and a similar symmetrical one) of the makespan function **ms** wich states when we can swap the jobs of a list while improving the makespan components:

  $$(\forall x\ \mathbf{in}\ l.\ x_1 \geq e_1) \wedge (\forall\ x\ \mathbf{in}\ l.\ x_2 \geq e_1) \wedge e_1 \leq e_2 \Rightarrow \mathbf{ms}([e] + p)_2 \leq \mathbf{ms}(p + [e])_2$$

  Then, (J3) is again obtained thanks to the invariant.

IV. A concurrent Johnson's procedure

Thanks to the invariant, we know that the schedule returned by Johnson's procedure is a 2-partition of the jobs. Then, based on this separation property, we propose the following parallelization:

$$\text{partition ; } (\text{sort}_1 \parallel \text{sort}_2) \text{ ; fusion}$$

where, we first partition the list of jobs, then *concurrently* we sort each partition according to the first and second component. Eventually, we fusion the sorted results. Since this solution validates the invariant and the correctness proofs rely on the invariant, we assert that this solution is correct by construction[2].

V. Conclusion

We have proposed a formal development of Johnson's procedure based on assertional reasoning. The invariant has been used for correctness purposes but also as a hint for parallelizing the procedure. Then, the correctness of the parallel procedure can be obtained for free. We believe that such an approach could be interesting for suggesting the parallelisations of other scheduling algorithms together with their correctness. For future work, we envision the formal validation of other scheduling algorithms, we are especially interested in realtime ones.

References

[1] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954. [Online]. Available: http://dx.doi.org/10.1002/nav.3800010110

[2] P. Brucker, *Scheduling Algorithms*, 3rd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

[2]More precisely, we have not looked, a posteriori, for its invariant.