An Optimizing Framework for Real-time Scheduling

Sakthivel Manikandan Sundharam, Sebastian Altmeyer, Nicolas Navet
University of Luxembourg
FSTC/Lassy
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
firstname.lastname@uni.lu

***Abstract*—Scheduling is crucial in real-time applications. For any real-time system, the desired scheduling policy can be selected based on the scheduling problem itself and the underlying system constraints. This paper discusses a novel optimization framework which automates the selection and configuration of the scheduling policy. The objective is to let designer state the permissible timing behavior of the system in a declarative manner. The system synthesis step involving both analysis and optimization then generates a scheduling solution which at runtime is enforced by the execution environment.**

I. Introduction

Real-time scheduling is now a mature and well established research field. Many scheduling policies and results have been proposed and derived over the last four decades providing efficient scheduling solutions for most hardware platforms and application-level needs. Tools and frameworks have been developed implementing these scheduling algorithms and their analyses [1]. To the best of our knowledge, apart from few early works in that direction for specific execution platforms (e.g. [8]), none of these frameworks target the automatic configuration and selection of the best suited policy and parameters in a systematic manner.

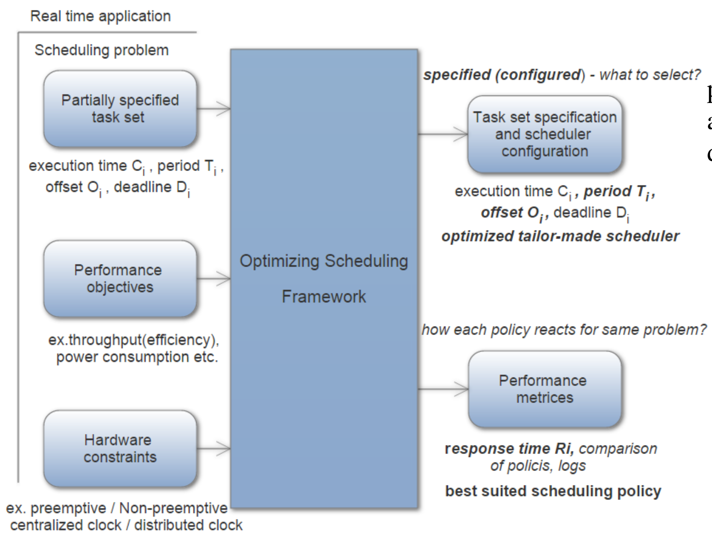
Real time application
Scheduling problem
Partially specified task set
execution time C_i , period T_i , offset O_i , deadline D_i
Performance objectives
ex.throughput(efficiency), power consumption etc.
Hardware constraints
ex. preemptive / Non-preemptive centralized clock / distributed clock
Optimizing Scheduling Framework
specified (configured) - what to select?
Task set specification and scheduler configuration
execution time C_i , period T_i, offset O_i, deadline D_i
optimized tailor-made scheduler
how each policy reacts for same problem?
Performance metrices
response time Ri, comparison of policis, logs
best suited scheduling policy

Fig. 1. Inputs/Outputs of the Framework

In this paper, we present an optimizing framework that selects the best suited scheduling configuration for a partially specified task set and the given system constraints. In Figure 1, we illustrate the structure of the framework. The inputs to the framework are:

- a partially specified task set (see Section II),
- the performance objectives, and
- the hardware constraints.

The framework performs the selection of the policy, optimization of the scheduling parameters, and outputs

- the complete task set specification and scheduling configuration,
- the performance metrics of the different scheduling algorithms.

This work is motivated by the ongoing research on timing-augmented Model-Based Design [7] at the University of Luxembourg. Our aim is to develop the framework such that the system designer only focuses on the high-level timing behavior of the system, where the implementation choices of the low-level timing behavior are taken care of by the framework. The framework fits in the early design phases as a device to automate system synthesis and hide away from the designer the complexity of the underlying runtime environments.

II. Defining The Framework

The inputs is the high-level description of the scheduling problem, whereas the output is the scheduling solution with all the required configuration parameters. In this section, we define inputs and outputs of our optimizing framework.

Partially specified task set:

We assume that the application is composed of n tasks $\{\tau_1, \ldots \tau_n\}$. The task set is defined partially to reflect the freedom in the selection of certain parameters. Each task τ_i is specified by a tuple

$$\tau_i : (C_i, T_i, D_i),$$

a) where C_i is the worst-case execution time and is assumed to be given as single value,
b) the execution period T_i is potentially defined as a range of permissible values,
c) the deadline relative to the release time of the task, denoted by D_i, is given as single value. A range of values for the deadline would be futile, as the run-time environment must ensure the system is feasible with respect to the most stringent deadline.

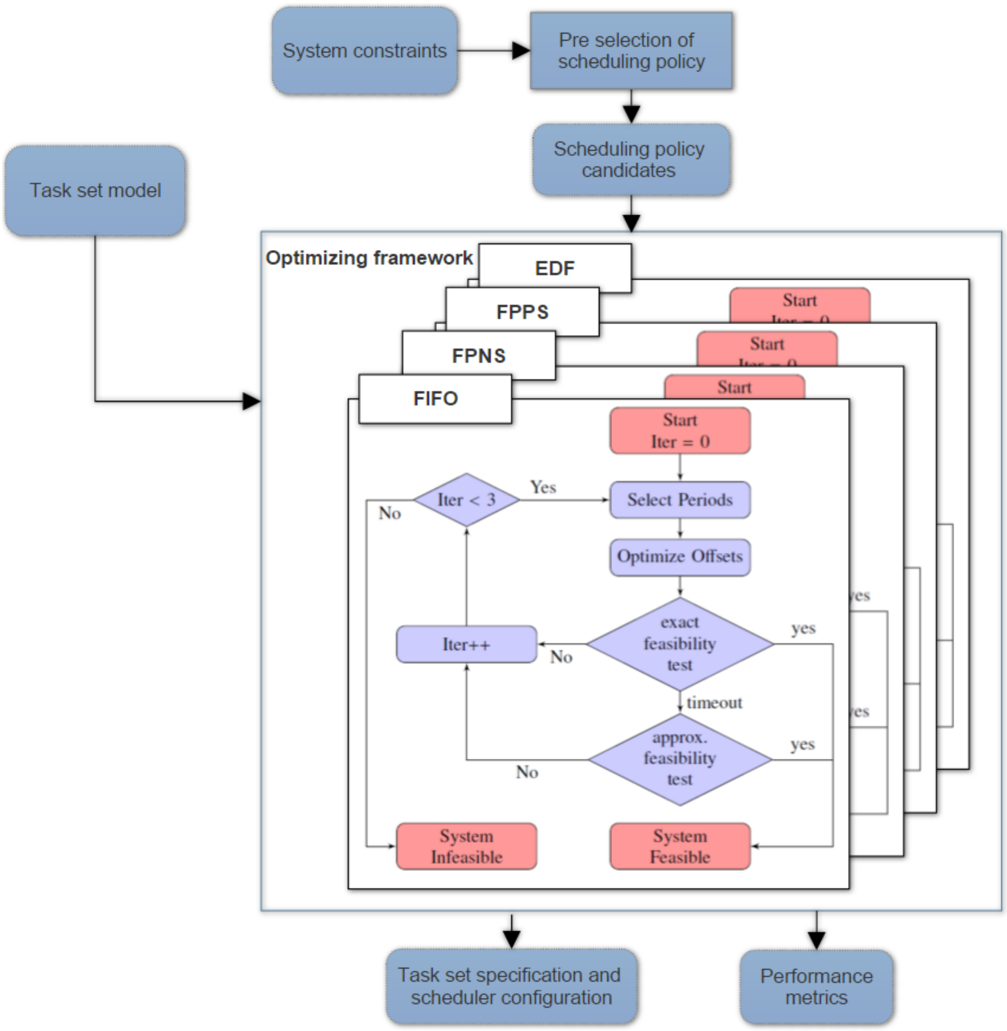

Fig. 2. Flow of the scheduler synthesis framework

Performance objectives:
Examples for performance objectives are throughput (efficiency), power consumption, predictability constraints such as requirement of uninterrupted execution for a task, minimal activation or end-of-execution jitters, etc. These objectives are achieved for example by minimizing the periods within the allowed range in order to reduce the power consumption. If for instance throughput is to be maximized, the frequency of execution is increased.

Hardware constraints:
The selected hardware further constraints the choices of the framework. Typical hardware, and more generally constraints of the run-time environment, are number of processing cores, preemptiveness, type of the system clock (e.g., global clock or distributed clock in the system). These inputs are accounted by the framework in the derivation of the scheduling solution.

Scheduling configuration:
The main outcome of the framework is the scheduling solution, that is the complete specification of the task set and the scheduling configuration. This scheduling configuration covers all low-level configuration parameters and no further input is needed to execute the application on the target system. In addition to the policy, scheduling parameters include periods, offsets and possibly deadlines and priorities.

Performance metrics:
The performance of the candidate algorithm is evaluated by selected performance metrics. Typical performance metrics are schedulability (a task set is schedulable or not under a policy), numerical values of the response times and jitters, power-consumption, or the ability of the system to grow further measured for instance by the minimum slack time.

III. Scheduler Synthesis

In this section, we explain the core of the framework, i.e., the scheduler synthesis. In contrast to other approaches towards scheduler synthesis [3], we do not generate new or non-standard scheduling policies. Instead, we focus on the selection of the most appropriate scheduling policy (including parameter optimization) amongst a set of well-studied and widely-implemented real-time scheduling policies. In Figure II, we illustrate the general steps of our framework.

In a first step, the framework performs a pre-selection of the scheduling policies on the basis of the system constraints and the hardware to execute the application. All policies that violate some of the requirements are excluded at this step. Policies that are compliant with the requirements under some side-constraints are considered with those side-constraints. This step results in the set of candidate policies which are subject to the actual parameter optimization.

The next step, the parameter optimization is then highly specific to the policy, and thus has to be performed for each policy individually. Also, the type of parameters to be optimized differ. However, we can build on a large variety of existing methods and techniques. For the selection of the periods, for instance, we can use a recent work by Nasri et al. [6], offsets in case of offset-aware policies can be optimized using [5] and for the selection of priorities, we have optimality results such as [2].

Real-time scheduling problems are in most contexts NP-hard. Due to the computational complexity of the problems, an optimal scheduling solution cannot be guaranteed. However, the candidate optimization techniques and heuristic algorithms have proven to be robust and lead to satisfactory solutions in many application domains (e.g., [5, 6, 8]).

IV. Illustrating Example

We illustrate our approach using the following task set Γ:

	C_i	T_i	D_i	constraints	objective
τ_1	1	[4 : 5]	4		reduce period
τ_2	2	[4 : 8]	8	non-preemptive	reduce period
τ_3	6	[15 : 24]	24	-	

Constraints and scheduling policies

A side constraint besides meeting deadlines is that task τ_2 has to be executed non-preemptively and the objective is to minimize the values of the periods of τ_1 and τ_2 (*i.e.*, increase frequency to achieve a better control of the system). To simplify the example, we restrain ourselves to a limited number of well-known scheduling policies: earliest deadline first (EDF), both preemptively and non-preemptively (EDF-NP), fixed-priority preemptive scheduling (FPP), fixed-priority non-preemptive scheduling (FPNP), and FIFO:

EDF	EDF-NP	FPP	FPNP	FIFO

The policy selection identifies that all policies can indeed satisfy the system constraints, but in case of the preemptive policies, i.e., EDF and FPP, further constraints are required to ensure the non-preemptive execution of τ_2:

EDF	EDF-NP	FPP	FPNP	FIFO
✓ if $D_2 \leq D_i$	✓	✓ if $pr_2 = 1$	✓	✓

All non-preemptive policies fail since the execution time of τ_3 exceeds even the largest permissible period of τ_1. Hence, the search has to concentrate only on the two remaining policies EDF and FPP (both with the appropriate side constraints).

EDF scheduling

Using EDF we are able to achieve a processor utilization of 1 and execute tasks τ_1 and τ_2 with the smallest possible periods. This is the optimal result and it will be selected as the scheduling solution with the following parameters for the task set:

	C_i	T_i	D_i
τ_1	1	4	4
τ_2	2	4	4
τ_3	5	20	20

For this particular scheduling problem — as well as in many other cases — EDF is the optimal solution. This situation changes when more complex side and system constraints are in place, or when we consider realistic scheduling overheads. The cache-related preemption delays (CRPDs) constitute an example of such overheads that, in this particular case, penalize preemptions. As a result, the advantages of EDF scheduling over FPPS often becomes negligible under CRPD overheads [4], As future work, we plan to include the modeling of these overheads, which will lead to other, less trivial optimal solutions.

V. Conclusions and Discussions

We are developing an optimizing framework that considers as inputs a partially specified task set, the performance objectives of the system and the constraints of the run-time environment, importantly the hardware support. The framework synthesizes the scheduling solution that best meet the requirements. Our framework currently includes a number of basic real-time scheduling policies and ongoing work is devoted to enrich the set of available policies with customized scheduler that make optimal use of underlying execution hardware and improvements in system behavior. This work is a contribution towards a more automated design process building on the wide set of techniques and results developed within the real-time system community.

The applicability and precision of the framework is determined by the optimization algorithms and schedulability analyses. But these algorithms and heuristic techniques are limited in precision. Consequently, for some scheduling problems, the framework cannot guarantee optimality. A future work is to develop techniques such as lower bounds to estimate how far is a solution computed with the framework from the optimal solution.

Acknowledgment

This research is supported by FNR (Fonds National de la Recherche), the Luxembourg National Research Fund (AFR Grant n°10053122).

References

[1] K. Altisen, G. Gossler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1999.

[2] N. C. Audsley. On priority asignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44, May 2001.

[3] M. Grenier and N. Navet. Fine tuning MAC level protocols for optimized real-time QoS. *IEEE Transactions on Industrial Informatics, special issue on Industrial Communication Systems*, 4(1), 2008.

[4] W. Lunniss, S. Altmeyer, and R. I. Davis. A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays. *Leibniz Transactions on Embedded Systems*, 1(1), 2014.

[5] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multisource software on multicore automotive ecus combining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, Oct 2012.

[6] M. Nasri and G. Fohler. An efficient method for assigning harmonic periods to hard real-time tasks with period ranges. In *27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, pages 149–159. IEEE, 2015.

[7] N. Navet, L. Fejoz, L. Havet, and S. Altmeyer. Lean model-driven development through model-interpretation: the CPAL design flow. In *Embedded Real-Time Software and Systems (ERTSS2016)*, January 2016.

[8] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm: Application to Posix1003.1b compliant OS. *IEE Proceedings Software*, 150(1):13–24, 2003.