Handling Criticality Mode Change in Time-Triggered Systems through Linear Programming

Mathieu Jan, Lilia Zaourar
CEA, LIST
Embedded Real Time Systems Laboratory
F-91191 Gif-sur-Yvette, France
Email: Firstname.Lastname@cea.fr

Vincent Legout
Virginia Tech
Blacksburg, Virginia, USA
Email: vlegout@vt.edu

Laurent Pautet
Institut Telecom
Telecom ParisTech
LTCI - UMR 5141 Paris, France
Email: Laurent.Pautet@telecom-paristech.fr

***Abstract*—Mixed Criticality helps reducing the impact of pessimistic evaluation of Worst Case Execution Time for real-time systems. This is achieved by hosting low-criticality tasks on a same hardware architecture in addition to the classical high-critical tasks, when considering two-criticality levels. The Time-Triggered paradigm (TT) is a classical approach within industry to develop high-criticality tasks. Extending TT systems in order to integrate the support of MC scheduling therefore requires the generation of two schedule tables, one for each criticality level. However, a switch between the schedule tables must not lead to an unschedulable situation for the high-criticality tasks. In this work, we show how a linear programming approach can be used to generate these schedule tables in a consistent way for dual-critical problems on multiprocessor architectures.**

I. Introduction

Industrial fields, such as automotive [4] or control automaton [7], consider the Time-Triggered [9] (TT) paradigm as a solution to build hard real-time systems. In the TT paradigm, the tasks are triggered by the advancement of time. The scheduling decisions are usually computed off-line and made available to the Real-Time Operating System (RTOS) through a schedule table. While the TT paradigm provides a predictable execution, the static scheduling approach is considered to lead to a poor resource utilization in the average case. The design of TT systems and the associated schedulability demonstrations must indeed be performed in the worst-case situation. These unused processing capabilities motivate the adding of Mixed-Criticality (MC) scheduling techniques within TT systems [1].

The goal of MC scheduling is to increase the schedulability of the low-criticality tasks, while still guaranteeing in the worst-case scenario the schedulability of the high-criticality tasks. In a previous work, we focused on the use of the elastic task model to include MC scheduling within TT systems [8]. In this work, we rely on the task model mainly used within the MC scheduling community [16], called the Vestal task model. This task model extends the classical periodic task model with: 1) two Worst-Case Execution Time (WCET) values, named $C_i(LO)$ and $C_i(HI)$, and 2) a criticality level χ_i, which can be either LO or HI. Then, two execution modes are assumed, namely HI and LO, and the system starts in the LO mode. $C_i(LO)$ is the maximum allowed execution time for the task in the LO mode, while $C_i(HI)$ is the maximum allowed execution time for the task in the HI mode. For the HI-criticality tasks we have $C_i(LO) < C_i(HI)$ and for the LO-criticality tasks $C_i(LO) = C_i(HI)$. The rationale is that the higher the criticality level is, the more conservative the verification process is and hence the greater the WCET value is. Whenever a HI-criticality task exceeds its assigned $C_i(LO)$ value, the system switches to the HI mode. In this mode, only the schedulability of the HI-criticality tasks is ensured, assuming $C_i(HI)$ for the WCET values.

Extending TT systems to cope with MC scheduling requires the definition of two schedules tables, named S_{LO} and S_{HI}. S_{LO} (resp. S_{HI}) is used while the system is in the LO (resp. HI) mode. [1] has stated the TT schedulability conditions that apply on these schedules for dual-criticality MC instances of task sets. The main issue is to guarantee that a mode change from S_{LO} to S_{HI} cannot lead to an unfeasible schedule for the HI-criticality tasks, i.e. the remaining time is not sufficient to completely schedule all the HI-criticality tasks. S_{HI} is indeed concerned by the schedulability of the HI-criticality tasks but should be built so that the schedulability of the LO-criticality tasks is maximized in S_{LO}. Building S_{LO} and S_{HI} cannot therefore be made independently in order to improve the resource utilization in the average case.

We propose two approaches to build S_{LO} and S_{HI} for (dual-criticality) instances of MC task sets. Both are based on the use of a linear programming approach. The remainder of this paper is as follows. Section II describes the related work. Section III formulates our linear programs to handle the criticality mode change in TT scheduling. Section VI provides a first analysis of our solutions and section VII concludes.

II. Related work

Note that existing work focuses on finite set of jobs whose exact arrival times are known a priori, as the results can be easily extended in order to address TT systems. The proposed algorithm must indeed only be applied over the hyper-period of the periodic task set being considered.

The first work on using MC scheduling within TT systems [1] studied how to generate S_{LO} and S_{HI} that can correctly schedule a MC job set modeled using the Vestal task model. It was inspired by the mode-change approach used to increase the flexibility of the TT scheduling in [5]. As the TT schedulability of MC tasks is NP-hard in the strong sense, they propose a polynomial-time algorithm for building S_{LO} and S_{HI} that is sufficient but not necessary. That is, the algorithm can fail to generate such tables for schedulable MC job sets, but if it succeeds then tables can correctly schedule them. The algorithm first computes a total priority ordering of the jobs using the Own-Criticality Based Priority (OBCP) algorithm [2]. Based on this priority ordering, S_{LO} is first built assuming $C_i(LO)$ for all the jobs. Then, S_{HI} is generated assuming this time $C_i(HI)$ for all the jobs (we remind that when $\chi = LO$, $C_i(LO) = C_i(HI)$).

[15] introduces a much more elaborated algorithm to build at the same time S_{LO} and S_{HI} assuming a slot scheduling approach. HI-criticality jobs are first splitted into two jobs noted J_i^{LO} and J_i^{Δ}. J_i^{LO} represents the $C_i(LO)$ of that job in the LO mode, while J_i^{Δ} represents the additional WCET assumed when in the HI mode (i.e. $\Delta_i = C_i(HI) - C_i(LO)$). Release time and deadline of these sub-jobs are computed so that each job has a maximum interval for its execution. A precedence constraint between sub-jobs is added in order to ensure a correct execution. Then, the proposed algorithm uses an heuristic to explore the set of possible scheduling decisions represented as a tree search. Based on the demand of HI-criticality jobs, a backtracking heuristic is used to cut from the tree search paths leading to unfeasible schedules.

[14] focuses on adding MC scheduling support within TT legacy systems, where the existing schedule table is considered to be S_{HI}. The proposed algorithm extends the slot-shifting based scheduling [6] in order to keep track of the spare capacities in each interval for both the HI and the LO-criticality jobs (named sc^{HI} and sc^{LO} respectively). A negative spare capacity means that some execution time must be borrowed from the other slots. If $sc^{LO} < 0$ then a HI-criticality job has exceeded its $C_i(LO)$ value and therefore only HI-criticality jobs must be executed. Finally, the legacy TT schedule is used only when $sc^{HI} = 0$. Note that this legacy TT schedule can prevent S_{LO} to be build, while [15] or [1] could produce a correct S_{LO}. It is the price to pay to avoid additional certification costs by keeping unchanged S_{HI}.

When considering TT systems, the previously introduced algorithms are called Single Time Table per Mode (STTM). [13] proves that the STTM approach dominates MC scheduling algorithms that define the priorities of jobs depending on the criticality mode (called FPM for Fixed Priority per Mode). They propose an algorithm in order to transform a FPM priority assignment into a set of STTM tables.

In this paper, the objectives of our contributions aim at revisiting these approaches for TT MC systems using Linear Programming (LP) techniques.

III. Problem Description using LP Approach

We first introduce the task model and notations we use in the remainder of this paper, before presenting our two linear programming approaches for building S_{LO} and S_{HI}.

As stated in the introduction, we rely on the Vestal task model and consider only two criticality levels, a restriction often assumed in MC scheduling [3]. We only state additional notations not introduced in the previous sections. Let $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ be a set of n independent, synchronous, preemptible and implicit deadline tasks. Tasks can migrate from one processor to another. We note M the number of processors. Each task $\tau_i \in \Gamma$ has the following temporal parameters $\tau_i = (\chi_i, P_i, C_i(LO), C_i(HI))$ with P_i the period of the task. Let H be the hyper-period of the task set. It is equal to the least common multiple of all periods of tasks in Γ. As in [11], the hyper-period H is divided in intervals, an interval being delimited by two task releases. We note n_{HI} the number of HI-criticality tasks and n_{LO} the number of LO-criticality tasks (thus $n_{HI} + n_{LO} = n$).

A job j can be present on several intervals and E_i is the set gathering these intervals. We note $w_{j,k}$ the weight of job j on interval k. We denote by I the set of intervals and $|I_k|$ the duration of the k^{th} interval. J_k is the set of jobs within interval k. The weight of each job is the amount of processor necessary to execute job j on interval $|I_k|$ only (it is not an execution time but a fraction of it). J_Γ is the job set of all jobs of Γ scheduled during the hyper-period H.

A job j_{LO} represents an instance of a LO-criticality task, while j_{HI} is an instance of a HI-criticality task. J_{LO} and J_{HI} are the job sets of respectively all the LO and the HI-criticality jobs from Γ. We note $w_{j,k}^{LO}$ the weight of job j in interval k in S_{LO}, while $w_{j,k}^{HI}$ is the weight of job j in interval k in S_{HI}. A RTOS is used to detect when a j_{HI}, i.e. HI-criticality job, exceeds its $C_i(LO)$ value.

Finally, note that our approach to build S_{HI} and S_{LO} is based on the use of LP to compute off-line the weights of each job on all intervals. Then, within an interval either a dynamic or static approach can be used to schedule jobs. As we are considering the TT approach, in this work we assume that the triggering of jobs is also computed off-line, for instance by using McNaugthon's algorithm [12]. These scheduling decisions are also stored in S_{HI} and S_{LO}. In this paper, we do not focus on this part of our scheduling approach.

IV. LP scheduling for HI-mode first: LPMC-HI

In our first proposal, S_{HI} and S_{LO} are built in two separate steps, although the objective used when building S_{HI} prepares the computation of S_{LO}. We express this solution as two linear programs, one for each table to build, and we name it LPMC-HI for *Linear Programming for Mixed-Criticality HI-mode first*. However, the constraints of the first linear program are the schedulability of J_{HI} tasks, while its objective is to optimize the schedulability of J_{LO} tasks in HI-mode. This prepares the input, i.e. the $w_{j,k}^{LO}$ for the jobs from J_{HI}, for the second linear programs dealing with the LO-mode.

First, we focus on building S_{HI}. The classical temporal schedulability constraints [11] are expressed to compute the optimal job weights on each interval for all the jobs from J_Γ. First, the sum of all job weights on an interval must not exceed the processor maximum capacity:

$$\sum_{j \in J_k} w_{j,k}^{HI} \leq M, \forall k \in I \tag{1}$$

Then, the weight of each job must not exceed each processor maximum capacity (no parallel jobs):

$$0 \leq w_{j,k}^{HI} \leq 1, \forall k \in I, \forall j \in J_\Gamma. \tag{2}$$

Finally, we express two different constraints for the completion of jobs from J_{LO} and J_{HI}. First, only the schedulability of the jobs from J_{HI} has to be ensured and therefore must be completely executed:

$$\sum_{k \in E_j} w_{j,k}^{HI} \times |I_k| = C_i(HI), \forall j \in J_{HI}. \tag{3}$$

Second, the schedulability of the jobs from J_{LO} may not be ensured while in S_{HI}. This means that some jobs from J_{LO} may not be completely executed:

$$\sum_{k \in E_j} w_{j,k}^{HI} \times |I_k| \leq C_i(LO), \forall j \in J_{LO}. \tag{4}$$

As far as S_{HI} is concerned, our objective is to prepare the building of S_{LO} in order to maximize the schedulability of J_{LO}, while still guaranteeing in the worst-case scenario the schedulability of J_{HI}. To this end, we introduce a decision variable to account when a job from J_{LO} has been completely executed, i.e. $\sum_{k \in E_j} w_{j,k}^{HI} \times |I_k| = C_i(LO)$. Let F_j be this decision variable that is equal to 1 if the job j from J_{LO} has been completely executed, and 0 otherwise. Then, our objective function can be defined as follows:

$$Maximize \sum_{j \in J_{LO}} F_j \tag{5}$$

This objective function computes the weights of a schedule in which a maximum number of jobs from J_{LO} are completely executed, while ensuring the schedulability for jobs from J_{HI}.

We now focus on the LO-mode. For building S_{LO} we have to compute $w_{j,k}^{LO}$ for each job from J_Γ assuming that its WCET is equal to $C_i(LO)$. That is the execution time of each job j from J_{HI} is reduced by Δ_i (see section II). For each job j in J_{HI}, $w_{j,k}^{LO}$ is equal to $w_{j,k}^{HI}$ till the $C_i(LO)$ is not exceeded. This differs from [1], where at a mode change, no scheduler could have given more time to the HI-criticality jobs than the proposed algorithm. In LPMC-HI, these jobs can be delayed in order to completely execute, over a set of intervals, some LO-criticality jobs.

Next, we have to compute $w_{j,k}^{LO}$ for all the jobs from J_{LO}. While S_{HI} was generated with a maximum number of jobs from J_{LO} completely executed, our second linear program could only compute the weights of the jobs from J_{LO} that have not yet been scheduled. However, we believe this reduces the search space when building S_{LO}, as we remind that only jobs from J_{HI} must be scheduled in S_{HI}. While a reduced search space seems an interesting property, as it decreases the execution time required for solving the linear program, we believe it also reduces the schedulability bound that can be achieved. To compute $w_{j,k}^{LO}$ for all the jobs from J_{LO}, the classical temporal constraints only have to be modified in order to use $w_{j,k}^{LO}$ for all the jobs from J_{HI} as fixed values and not as variables. In the next equation, the value of a variable w is noted w' to depict this point :

$$\sum_{j_{LO} \in J_k} w_{j,k}^{LO} + \sum_{j_{HI} \in J_k} w_{j,k}^{LO'} \le M, \forall k \in I \tag{6}$$

$$0 \le w_{j,k}^{LO} \le 1, \forall k \in I, \forall j \in J_{LO}. \tag{7}$$

$$\sum_{k \in E_j} w_{j,k}^{LO} \times |I_k| = C_i(LO), \forall j \in J_{LO}. \tag{8}$$

Note that this second LP has no objective function as any feasible solution given by the solver generates a valid scheduling.

LPMC-HI can lead to situations where S_{LO} cannot be computed. The jobs from J_{HI} are indeed concentrated in some particular intervals in S_{HI} and then their total weights are simply reduced over these intervals to match their lower $C_i(LO)$. However, redistributing the weights of jobs from J_{HI} while computing S_{LO} would increase the schedulability bound that can be achieved for the jobs from J_{LO}. Section VI illustrates this point using an example.

V. LP scheduling for both LO- and HI-modes: LPMC-Both

In our second approach, we explore such an alternative strategy for computing the weights in order to improve the success ratio of the scheduling. We thus consider the generation of S_{LO} and S_{HI} at the same time, i.e. within the same linear program, and therefore name this approach *LPMC-Both*. We split each HI-criticality job into two sub-jobs: j_{LO} and j_Δ and consider j_{LO} as a LO-criticality job that we added in J_{LO}. A precedence constraint will be expressed later to ensure building correct schedules. LPMC-Both is similar to [15] and we therefore use the same notations as in this work (see sect. II). Additionally, we note $w_{j,k}^{\Delta}$ the weight of a job j_Δ.

A first set of constraints must be expressed for S_{HI} in order to ensure the schedulability of all the jobs from J_{HI}. This is similar to the equations (1), (2) and (3). The only difference is that now the weight of each job in S_{HI} is defined as follows:

$$w_{j,k}^{HI} = w_{j,k}^{LO} + w_{j,k}^{\Delta}, \forall k \in I, \forall j \in J_{HI} \tag{9}$$

Precedence constraints are then required to ensure a correct schedule of each job from J_{HI}, that is the $w_{j,k}^{\Delta}$ must be null till $\sum_{m=1}^{k} w_{j,m}^{LO} \times |I_m| \le C_i(LO)$. This prevents sub-jobs j_{LO} and j_Δ to be present in the same interval in S_{LO}. Avoiding this situation ensures that a criticality mode change from S_{LO} to S_{HI} is possible, i.e. that it does not lead to an unfeasible schedule, at every point where all the jobs from J_{HI} can first exceed their $C_i(LO)$ values. This corresponds to the *switch through property* described in [5] for these points. Note that this property is ensured in our first scheduling approach by how we compute $w_{j,k}^{LO}$ for each job j from J_{HI}. In the first interval k in which a job j_{HI} exceeds its $C_i(LO)$ value, note that the two sub-jobs j_{LO} and j_Δ can be present. However, as $w_{j,k}^{HI}$ cannot be higher than 1 (eq. 2), the weight left to j_Δ in interval k is constrained so that a schedule where j_{LO} and j_Δ cannot be executed in parallel can be found (i.e $w_{j,k}^{\Delta} + w_{j,k}^{LO} \le |I_k|$). Finally, in the other intervals the solver has no constraint for computing $w_{j,k}^{\Delta}$.

Then, a second set of constraints must be expressed for S_{LO} in order to ensure the schedulability of all the jobs from J_{LO}. These constraints are identical to the equations (7) and (8), in addition to the following constraint:

$$\sum_{j \in J_k} w_{j,k}^{LO} \le M, \forall k \in I \tag{10}$$

Finally, we use the same objective function as (5), that is maximize the number of schedulable jobs from J_{LO}. It therefore requires the same decision variable to account when a job from J_{LO} has been completely executed. In the end, if a solution can be found, then the output of LPMC-Both is the weights of each job to be used to build both S_{LO} and S_{HI}.

VI. First analysis of LPMC-HI and LPMC-Both

We first compare LPMC-HI and LPMC-Both in terms of complexity. We first focus on LPMC-HI. The total number of decision variables in the first LP of LPMC-HI is equal to $|I| \times n$ for the weights of all jobs, plus $|J_{LO}|$ for the F_j. In the second LP of LPMC-HI, this number is reduced to $|I| \times n_{LO}$ as only the weights of jobs from J_{LO} are computed when building S_{LO}. In the first (resp. second) LP of LPMC-HI, the number of constraints is equal to its number of variables plus $|I| + |J_\Gamma|$ (resp. $|I| + |J_{LO}|$) due to the equations (1), (3) and (4) (resp. (6) and (8)). We now focus on LPMC-Both. Compared to LPMC-HI, the total number of decision variables in LPMC-Both is increased by $2 \times |I| \times n_{HI}$. This comes from additional weights introduced by the job splitting and for implementing the precedence constraints. The number of constraints of LPMC-Both is equal to the sum of: $|I| \times (n+1) + |J_\Gamma|$ for computing S_{LO}, $|I| \times (n_{HI}+1) + |J_{HI}|$ for computing S_{HI} and $2 \times |I| \times n_{HI}$

	χ_i	P_i	$C_i(LO)$	$C_i(HI)$
τ_1	LO	2	1.5	1.5
τ_2	HI	4	2	3
τ_3	HI	3	1	2

TABLE I. TASK SET WITH τ_1 A LO-CRITICALITY TASK.

for dealing with the precedence constraints. The complexity of LPMC-Both is therefore higher than the complexity of LPMC-HI. However, the computational complexity of LPMC-HI and LPMC-Both depends on the number of intervals, which is limited in industrial configurations usually showing harmonic periods ([7], [4]).

We now compare both approaches in terms of efficiency. Table I depicts a task set running on a dual-core ($M = 2$) and made of three tasks where τ_1 is a LO-criticality task. Figure 1 shows S_{HI} computed by LPMC-HI. The third and sixth instances of τ_1 cannot be completely executed in the intervals I_4 and I_8, leading to $F_1 = 4$ (out of 6). Note that the second and fifth instances of τ_1 span over 2 intervals, i.e. respectively I_2, I_3 and I_6, I_7. The other instances require only 1 interval. When trying to compute the corresponding S_{LO}, $w_{3,4}^{LO}$ is equal to 0.5, as the C_i of the second instance of τ_3 is reduced by 1 unit of time in interval I_4. However, the third instance of τ_1 cannot be scheduled as $w_{1,4}^{LO}$ should be equal to 0.75 in order to satisfy the equation (8). But then, the equation (6) would not be satisfied as the utilization would be equal to 2.5 and hence higher than M. A valid S_{LO} cannot therefore computed. As shown by Figure 2, both S_{LO} and S_{HI} can be computed using LPMC-Both thanks to its ability to distribute the weights over all the intervals.

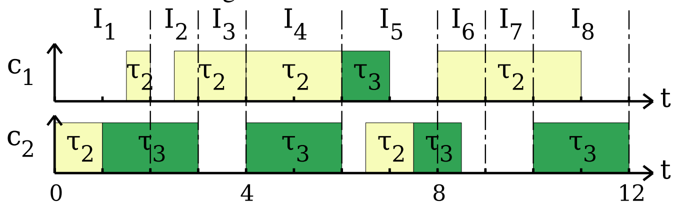

Fig. 1. Possible S_{HI} for the task set of table I computed by the LPMC-HI leading to an unfeasible S_{LO}.

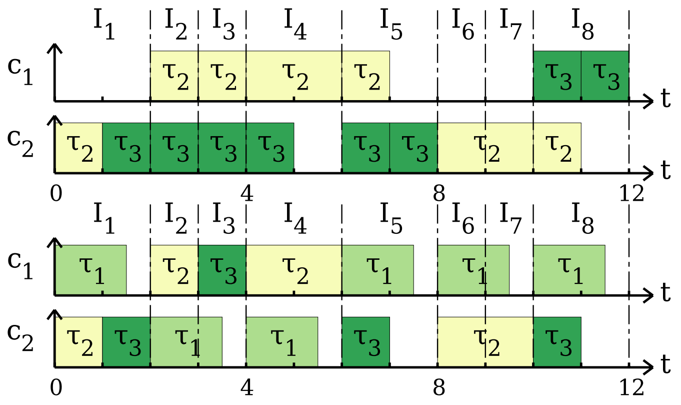

Fig. 2. Possible S_{HI} (top) and S_{LO} (bottom) for the task set of table I computed by LPMC-Both.

VII. CONCLUSION

The Time-Triggered (TT) paradigm is one solution used within industrial fields to design hard real-time systems subject to certification constraints. While the TT paradigm provides interesting properties, such as determinism, this comes at the price of low resource utilization in the average case. Mixed-Criticality (MC) scheduling aims at providing an efficient use of the processing capabilities available in the average case through the execution of low-criticality tasks, while ensuring schedulability for the high-criticality in the worst-case.

TT relies on a off-line computation of scheduling decisions made available at run-time through a schedule table. In this work, we consider dual-critical problems requiring the construction of two schedule tables. The main difficulty when building them is to ensure that switching from the LO table to the HI table is possible, i.e. does not lead to unfeasible schedules when a HI criticality task exceeds its LO behaviour. We propose two approaches, named LPMC-HI and LPMC-Both, based on the use of linear programs to build these tables. We are currently implementing them in order to evaluate their success ratio in scheduling of job sets whose utilizations are uniformly distributed, as in [15]. In future work, we plan to integrate additional constraints in the generation of TT schedule, such as energy consumption as presented in [10].

REFERENCES

[1] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Proc. of the 32nd Real-Time Systems Symp. (RTSS)*, pages 3–12, Vienna, Austria, Nov 2011.

[2] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proc. of the 16th Intl. Conf. on Real-Time and Embedded Technology and Applications Symp. (RTAS)*, pages 13–22, Stockholm, Sweden, April 2010.

[3] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Trans. Computers*, 61(8):1140–1152, 2012.

[4] D. Chabrol, D. Roux, V. David, M. Jan, M. A. Hmid, P. Oudin, and G. Zeppa. Time- and angle-triggered real-time kernel for powertrain applications. In *Proc. of the Design, Automation & Test in Europe Conf. (DATE)*, pages 1060–1063, Grenoble, France, March 2013.

[5] G. Fohler. Changing operational modes in the contex of pre run-time scheduling. E76-D(11):1333–1340, 1993.

[6] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proc. of the 16th Real-Time Systems Symp. (RTSS)*, pages 152–161, Pisa, Italy, Dec. 1995.

[7] M. Jan, V. David, J. Lalande, and M. Pitel. Usage of the safety-oriented real-time OASIS approach to build deterministic protection relays. In *Proc. of the 5^{th} Intl. Symp. on Industrial Embedded Systems (SIES 2010)*, pages 128–135, Trento, Italy, July 2010.

[8] M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low-criticality tasks in mixed criticality systems. In *Proc. of the 1st Intl. Workshop on Mixed Criticality Systems (WMC)*, pages 43–48, Vancouver, Canada, December 2013.

[9] H. Kopetz. The time-triggered model of computation. In *Proc. of the Real-Time Systems Symp. (RTSS)*, pages 168–177, Madrid, Spain, 1998.

[10] V. Legout, M. Jan, and L. Pautet. Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses. In *Proc. 1st workshop on Real-Time Mixed Criticality Systems*, Taiwan, Aug. 2013.

[11] M. Lemerre, V. David, C. Aussaguès, and G. Vidal-Naquet. Equivalence between schedule representations: Theory and applications. In *Proc. of the Real-Time and Embedded Technology and Applications Symp.*, pages 237–247, St. Louis, USA, 2008.

[12] R. McNaugthon. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, October 1959.

[13] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Time-triggered mixed-critical scheduler. In *Proc. of the 1st Intl. Workshop on Mixed Criticality Systems*, pages 67–72, Vancouver, Canada, Dec. 2013.

[14] J. Theis and G. Fohler. Mixed criticality scheduling in time-triggered legacy systems. In *Proc. of the 1st Intl. Workshop on Mixed Criticality Systems (WMC)*, pages 73–78, Vancouver, Canada, December 2013.

[15] J. Theis, G. Fohler, and S. Baruah. Schedule table generation from time-triggered mixed criticality systems. In *Proc. of the 1st Intl. Workshop on Mixed Criticality Systems (WMC)*, pages 79–84, Vancouver, Canada, December 2013.

[16] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the 28th Real-Time Systems Symp. (RTSS)*, pages 239–243, Tucson, USA, Dec. 2007.