Schedule Table Generation for Time-Triggered Mixed Criticality Systems

Jens Theis and Gerhard Fohler
Technische Universität Kaiserslautern, Germany

Sanjoy Baruah
The University of North Carolina, Chapel Hill, NC, USA.

***Abstract*—Recent research in real-time scheduling for mixed criticality systems has centered on an event-triggered/priority-driven approach to scheduling. Current practice in many safety-critical domains, however, favors a time-triggered approach.**

We present here an effective and flexible approach for applying mode changes for time-triggered systems to handle mixed criticality job sets.

It is based on a heuristic search algorithm for constructing schedule tables for the different criticalities: a change of criticality levels results in a change in the schedule table that is being used. The sequence of steps in case of a change of criticality level is known beforehand.

We present a search-tree based framework for our heuristic search, and derive a heuristic function for guiding the search that significantly reduces the search space for backtracking by swapping search decisions over several levels in the search tree.

I. Introduction

The common approach to design embedded real-time systems with safety critical requirements, subject to certification, has been complete spatial and temporal isolation between activities in the system (e.g., in the ARINC standard [1] for avionics). In many modern platforms, however, the impact on performance and resource utilization of such strict separation approaches can no longer be justified by certification efforts, even more so as over pessimistic assumptions are mandated.

In order to certify a system as being correct, the certification authorities (CAs) mandate certain assumptions about the worst-case behavior of the system during run-time; these assumptions, e.g., for execution time, are typically far more conservative than the assumptions that the system designer would use during the process of designing, implementing, and testing the system if subsequent certification were not required. However, while the CAs are only concerned with the correctness of the safety-critical part of the system, the system designer is responsible for ensuring that the entire system is working correctly, including the non-critical parts. Traditional scheduling criteria as deadlines, utilization, etc. have proved inadequate for accommodating these contrasting demands of low and high critical applications.

In the time-triggered (TT) paradigm [13] of real-time scheduling, activities in the system are triggered by the progression of time only. A schedule for the entire duration of a system's execution is constructed prior to run-time. The scheduling decision that is made at each instant during run-time is completely determined by examining this pre-computed schedule, represented, e.g., in a schedule table. The schedule tables typically used for TT systems are particularly easy to verify; hence, they have been popular in safety critical systems subject to certification.

In mixed criticality systems as described above, however, the inflexibility of TT scheduling poses additional challenges: tasks with different assumptions cannot be fit into a single schedule table or changed, should the need arise during system operations.

While current practice in many safety-critical application domains is centered on time-triggered (TT) scheduling, much recent research on scheduling for mixed criticality systems has focused on even-triggered/priority-based scheduling.

In [5], an effort was made to show that the results obtained by this mixed criticality research could indeed be applied to TT-scheduled systems. It advocated the creation of two different schedule tables, one based upon the system designers' job parameters and the other using the CAs' parameters. At run-time the system starts operation with the schedule table that was constructed assuming that the system designers' parameters are correct. If a violation of the designer assumptions is detected during run-time, the run-time dispatcher switches tables and henceforth begins using the schedule table that was constructed assuming that the CAs' parameters are the correct ones.

Challenges in the construction of these schedule tables include the need to build two matching schedule tables, which allow for feasible and consistent switching from one to the other at run-time while ensuring, e.g., that even during a switch, the computational demands of individual jobs that are active during the switch are met. Since jobs with two different WCET values, one for each criticality level (mode), have to be considered, standard mode change schedule-generation algorithms (such as the ones presented in [11], [12]) cannot be directly applied. The simple table-generation algorithm of [5] was based on identifying a common priority ordering of the jobs for both schedules, with the priority of each job being based on its criticality level. This priority ordering is inflexible, and the resulting resource utilization is less than may be achieved by dynamic scheduling approaches such as EDF.

We believe that [5] was to a large measure successful in the sense that (i) mixed criticality scheduling principles were transferred to the TT domain; (ii) a TT-based framework for implementing mixed criticality systems was designed; and (iii) proof-of-concept algorithms for generating the schedule tables needed by this framework were derived. However, TT scheduling typically uses very sophisticated search-based algo-

This work has been supported in part by the European project DREAMS under project No. 610640.

rithms for schedule table generation, thereby achieving good resource utilization in practice despite the poor worst case bounds that can be derived for pathological workload instances. Hence, a true integration of TT and mixed criticality scheduling requires that it must be shown that schedule tables can be generated that are far more resource-efficient than the ones generated by the method in [5].

In this paper, we present an algorithm for constructing the schedule tables and for run-time execution that results in far more efficient resource utilization and enhanced flexibility; in contrast to the highly simplified proof-of-concept tables obtained by the techniques of [5], these tables can be used for actual system implementation. We have developed a tree-based search algorithm based on a heuristic function called the *leeway* (described in Section V), and a sophisticated and innovative backtracking algorithm based on this heuristic, that we believe shows this.

Related work. It is beyond the scope of this paper to discuss all work on mixed criticality scheduling; instead, we refer the interested reader to the recent survey by Burns and Davis [9].

In [4], Baruah et al. proved that mixed criticality scheduling is NP-hard in the strong sense even for two criticality levels — this provides some justification for seeking heuristic approaches to schedule construction. Further, [4] considered two classes of mixed criticality scheduling algorithms. Reservation-based scheduling policy, e.g., the straightforward approach by assigning $WCET := max\,(WCET(\chi_i))$ for all criticality levels χ_i, is a very pessimistic approach. Depending on the difference between the minimum and maximum WCET of the criticality levels, huge differences between the reserved WCET and the WCET of the actual criticality level are possible. Hence, this leads to an under-utilization of the system. The advantage of this approach is low complexity schedulability test. Additionally [4] considered the class of priority-based scheduling policies. As an example, they used the "Audsley-approach" (see [2], [3]) of assigning priorities; this assignment of fixed priorities to jobs compromises resource utilization and reduces flexibility.

In [10], de Niz et al. presented a preemptive, priority based algorithm. In this algorithm, high criticality tasks are protected from interference by low criticality tasks. This Rate-Monotonic-based algorithm determines the point in time when the priority has to be increased such that the deadline can be met with the high criticality WCET.

Park and Kim presented in [15] an online algorithm based on EDF for scheduling mixed criticality jobs. Based on the deadlines of the jobs, they created intervals and calculated slacks for these intervals. An earlier completion of a high criticality job, i.e., actual execution is less than WCET, increases the "remaining slack". Further, the available time when all high criticality jobs are guaranteed with high criticality WCET is considered as so-called "empty slack". Low criticality jobs are only executed if "remaining slack" or "empty slack" is greater than zero.

Organization. The remainder of the paper is structured as follows: in Section II, we present terms and the basic task model. Section III shows the basic idea of handling mixed criticality jobs based on our time-triggered approach with mode changes. The allocation of jobs to modes, depending on their criticality levels, is shown in Section IV. In Section V, we show the algorithm to construct the time-triggered schedule tables. We discuss our backtracking heuristic and the consequences for the scheduling process in Section VI. The evaluation of our methods is shown in Section VII. Finally, Section VIII concludes the paper.

II. Terminology and Notation

In the following, we present the terms that we use in the rest of the paper. We assume a system with two criticalities: low (LO) and high (HI). Unless otherwise specified we will represent relative time values (e.g. WCETs) by using upper case variables, whereas lower case variables represent absolute time values (e.g. release times). The dual criticality jobs J_i with $i \in \{1,..,n\}$ are characterized by the 5-tuple $\langle \chi_i, r_i, d_i, C_i(\text{LO}), C_i(\text{HI})\rangle$, with $\chi_i \in \{\text{LO}, \text{HI}\}$: criticality level of job i, $r_i \in \mathbb{R}^+$: release time of job i, $d_i \in \mathbb{R}^+$: absolute deadline of job i with $d_i > r_i$, $C_i(\text{LO})$: LO criticality WCET (as estimated by system designer), and $C_i(\text{HI})$: HI criticality WCET (as estimated by certification authority).

For LO criticality jobs, we assume $C_i(\text{LO}) = C_i(\text{HI})$, whereas for HI criticality jobs $C_i(\text{LO}) \leq C_i(\text{HI})$. Note that the assumption that $C_i(\text{LO}) = C_i(\text{HI})$ for low criticality jobs supposes the presence of run-time mechanisms for monitoring the amount of time that a job has executed, and preventing a job from executing beyond an allocated "budget". Such mechanisms are commonly found in most real-time operating systems. A low criticality job would then be allocated an execution budget equal to its LO criticality WCET, its $C_i(\text{LO})$ value.

The demand $g(t_1, t_2)$ in an interval $[t_1, t_2]$ is defined as the amount of processing time requested by jobs that are activated in that interval. The demand calculation considers all jobs whose release time is after the beginning of the interval ($r_i \geq t_1$) and must be completed before the end of the interval ($d_i \leq t_2$). It is known that it is sufficient to check the intervals from zero to the latest deadline in the system [6].

III. TT Systems with Mode Changes

In this section, we state the requirements placed on the scheduling of mixed criticality job sets and how we can accommodate these requirements within the framework of a time-triggered system. HI criticality jobs are subject of certification under pessimistic assumptions of the CAs. Furthermore, the designer has to ensure that the entire job set is feasible under his less pessimistic assumptions. Time-triggered schedule tables allow to simplify the certification process by their complete determinism. A feasible schedule table represents a constructive proof of timing correctness. The disadvantage of schedule tables is their inflexibility. Baruah and Fohler showed in [5] that constructing one schedule table per criticality level can fulfill CAs' requirements. The challenge in the construction of the schedule tables is to meet the requirements of mixed

original jobs					split jobs				
job	release time	deadline	WCETs	criticality	job	release time	deadline	WCET	criticality
J_i	r_i	d_i	$C_i(\text{LO}) = C_i(\text{HI})$	LO	J_i	r_i	d_i	$C_i(\text{LO})$	LO
J_i	r_i	d_i	$C_i(\text{LO}) \leq C_i(\text{HI})$	HI	J_i^{LO}	r_i^{LO}	d_i^{LO}	$C_i^{\text{LO}}(\text{LO})$	HI
					J_i^{Δ}	r_i^{Δ}	d_i^{Δ}	$C_i^{\Delta}(\text{HI})$	HI

TABLE I
OVERVIEW OF JOB PARAMETERS

criticality jobs and guaranteeing CAs' requirements when switching between the two schedule tables. A possible solution for this can be accomplished via the mechanism of *mode changes*. By the use of mode change schedulers (e.g. [11]), we can accommodate for the afore-mentioned demands. [5] describes the requirements for a mode-change-based scheduler for certification-cognizant mixed criticality jobs as a proof of concept. Two modes are used, one for each criticality level, to accommodate for the requirements of LO and HI criticality behavior. In the LO criticality mode LO-*table*, correct system behavior for the entire job set is guaranteed, based on the system designer's assumptions. Assumptions of the CAs are incorporated into the HI criticality mode HI-*table*. Note that for the purposes of certification it is sufficient to guarantee the correct execution of only the HI criticality jobs, but these must be guaranteed based on the pessimistic assumptions about the WCET of the CAs. However, the table-generation technique presented in [5] includes all jobs, both the LO-criticality and the HI-criticality ones, in the HI-*table*; this leads to an under-utilization of system resources.

We will construct two schedule tables, one for LO and one for HI criticality mode, such that switching from LO-*table* to HI-*table* is possible at every point in time, so-called *switch through property* [11]. At run-time, the system starts execution of the LO criticality schedule table. Violation of the system designer's assumptions (exceeding $C_i(\text{LO})$ of a HI criticality job) leads to directly switching to the HI criticality mode HI-*table*. In case such a switch occurs, we must guarantee that no HI criticality job misses its deadline. Furthermore, the CAs' pessimistic assumptions must be guaranteed for HI criticality jobs. According to the problem statement in [5], switching back to the LO criticality mode is not specified.

IV. Allocation of Jobs to Modes

Our method works on jobs, hence, it is also able to handle task sets as each individual task is composed of a sequence of recurring jobs. In this section, we describe how we split the jobs to separate the WCET of the LO criticality level and the additionally needed WCET in HI criticality case. After doing so, each resulting job is specified only with a single WCET.

LO criticality jobs J_i, which only are present in LO-*table*, are not split and the WCET is set to $C_i(\text{LO})$. The remaining parameters of these jobs remain unchanged. Each HI criticality job J_i is split into a job J_i^{LO}, which is the portion present in both tables and a job J_i^{Δ}, which is the portion that is additionally needed in HI criticality case. The calculation of the WCETs, which we will use as input for our scheduler, are shown in equations (1) - (3).

$$J_i : C_i(\text{LO}) \leftarrow C_i(\text{LO}) \,,\; i \in \{1,..,n\} \wedge \chi_i = \text{LO} \tag{1}$$

$$J_i^{\text{LO}} : C_i^{\text{LO}}(\text{LO}) \leftarrow C_i(\text{LO}) \,,\; i \in \{1,..,n\} \wedge \chi_i = \text{HI} \tag{2}$$

$$J_i^{\Delta} : C_i^{\Delta}(\text{HI}) \leftarrow C_i(\text{HI}) - C_i(\text{LO}) \,,\; i \in \{1,..,n\} \wedge \chi_i = \text{HI} \tag{3}$$

For the split jobs we derive now new parameters based on the original parameters of the HI criticality job. The release time r_i^{LO} of J_i^{LO} is equal to original release time r_i. The deadline of J_i^{LO} must be early enough such that there remains enough time to schedule the additionally needed WCET in the HI criticality case. As a result, the deadline of J_i^{LO} is set to $d_i^{\text{LO}} := d_i - C_i^{\Delta}(\text{HI}) = d_i - (C_i(\text{HI}) - C_i(\text{LO}))$.
The earliest release time of J_i^{Δ} is possible when its corresponding job J_i^{LO} is scheduled directly at the beginning of the execution window. Hence, we set the release time of a job J_i^{Δ} to $r_i^{\Delta} := r_i + C_i^{\text{LO}}(\text{LO}) = r_i + C_i(\text{LO})$. The deadline of J_i^{Δ} is equal to the deadline of its corresponding original job $d_i^{\Delta} := d_i$. This results in maximum slack for both J_i^{LO} and J_i^{Δ}. To avoid that J_i^{Δ} is scheduled before J_i^{LO}, ***we add a precedence constraint*** between them: $J_i^{\text{LO}} \prec J_i^{\Delta}$. As a consequence, we can guarantee with J_i^{Δ} that in HI criticality case the additionally needed WCET is scheduled. The two resulting jobs (J_i^{LO} and J_i^{Δ}) of a split HI criticality job (J_i) keep their criticality level HI. Table I gives an overview of the original and the split jobs' parameters.

The LO criticality table LO-*table* contains all the jobs in the set $S(\text{LO})$ while the HI criticality table HI-*table* contains all the jobs in the set $S(\text{HI})$ with:

$$S(\text{LO}) = \{J_i, J_k^{\text{LO}}\} \,, \quad (i \in \{1,..,n\} \wedge \chi_i = \text{LO}) \wedge (k \in \{1,..,n\} \wedge \chi_k = \text{HI}) \tag{4a}$$

$$S(\text{HI}) = \{J_i^{\text{LO}}, J_i^{\Delta}\} \,, \quad (i \in \{1,..,n\} \wedge \chi_i = \text{HI}) \tag{4b}$$

V. Construction of the Scheduling Table

The schedule table is divided into slots, which means that this is the granularity of our scheduler, hence, it is preemptive at slot borders. Construction of the scheduling tables is done concurrently for both tables, i.e. first slot i – which refers to the time interval $[i, i+1)$ – is scheduled in both tables (first LO-*table*, then HI-*table*), before in the next step slot $(i+1)$ is scheduled. The length of the schedule table is determined by the last deadline of the job set or the hyper-period in case of a periodic task set. Scheduling decisions are represented by a *search tree* which is based on iterative deepening [14]. Each scheduling decision for a slot in both tables (i.e. a pair of selected jobs) is represented by an edge in the search tree. Based on the history of decisions a node represents a possible partial schedule of

both tables at a given point in time. In each node of a path, several scheduling decisions can be taken, leading to different (partial) schedules. All combinations of possible scheduling decisions form the complete search tree. Leaf nodes represent feasible and infeasible complete schedules. The selection of jobs in both tables uses a heuristic which is based on EDF and the criticality levels. If a scheduling decision leads to infeasible schedule tables, we use backtracking to search for another schedule table. In "classic" backtracking, an exhaustive search checks all possible decisions in a search tree which is extremely complex. We use a heuristic for backtracking which is based on the demand of HI criticality jobs to reduce the complexity of backtracking.

A. Low Criticality Scheduling Decisions

In the LO criticality mode LO-*table*, we select the job of the set $S(\text{LO})$ with the earliest deadline which is ready.

We introduce the concept of the *leeway* $\delta(s_c)$ of the current slot s_c, see equation (5), as a heuristic function for our backtracking mechanism. The calculation of the leeway depends on the criticality level of the selected job: for LO criticality jobs J_i, the leeway represents the difference between the deadline of the selected job and the current time (i.e. end of current slot). For HI criticality jobs J_i^{LO}, the leeway represents the difference between the deadline of the selected job (J_i^{LO}) and the current point in time ***reduced*** by the remaining demand of all J_k^{Δ} with $k \in \{1, .., n\}$ which have to be scheduled in HI-*table* until the deadline of J_i^{Δ}. The demand $g^{\Delta}(t)$ at time t represents the demand in the interval $[0, t]$ accumulated by all jobs J_i^{Δ}. The function $g^{\Delta}_{sched}(t)$ keeps track of the already scheduled demand of HI criticality jobs J_i^{Δ}. If there is no job scheduled in a slot, we define the leeway to be infinity.

$$\delta(s_c) = \begin{cases} d_i - (s_c + 1) & \text{if } \chi_i = \text{LO} \\ [d_i - (s_c + 1)] & \\ \quad - \left[g^{\Delta}(d_i^{\Delta}) - g^{\Delta}_{sched}(s_c)\right] & \text{if } \chi_i = \text{HI} \\ \infty & \text{else} \end{cases} \tag{5}$$

Slot s_c in the HI criticality schedule table HI-*table* has not been scheduled yet, hence, $g^{\Delta}_{sched}(s_c)$ does not include the scheduled demand of HI criticality jobs J_i^{Δ} in the current slot in HI-*table*. For each slot in the mode LO-*table*, we calculate the leeway. A non-negative leeway means it can be possible to schedule remaining jobs J_i^{Δ} in the HI criticality mode. Thus, we continue the scheduling process by scheduling the current slot in mode HI-*table*. If the leeway is negative, then independent of succeeding scheduling decisions, all paths will lead to leaves representing infeasible schedules. As a consequence, we start backtracking based on our heuristic (see section VI). Based on the heuristic function (leeway), a preceding node is searched for backtracking, i.e., we change the scheduling decision for that slot.

B. High Criticality Scheduling Decisions

Based on the decision in LO-*table*, we select a job for mode HI-*table*. If the scheduled job in the current slot in mode LO-*table* has criticality level HI, i.e. J_i^{LO}, then we schedule the same job J_i^{LO} in the current slot in mode HI-*table*. If the scheduled job is a LO criticality job J_i (or no job is scheduled) then we select a HI criticality job J_k^{Δ} with fulfilled precedence constraints and earliest deadline. After scheduling a HI criticality job J_i^{Δ}, we increase the amount of already-scheduled demand $g^{\Delta}_{sched}(s_c)$ by one slot. After scheduling the current slot, we check whether a deadline miss of a HI criticality job J_i^{Δ}. occurred. In this case, the scheduling process is aborted. After scheduling the HI criticality mode HI-*table*, we continue with scheduling the next slot.

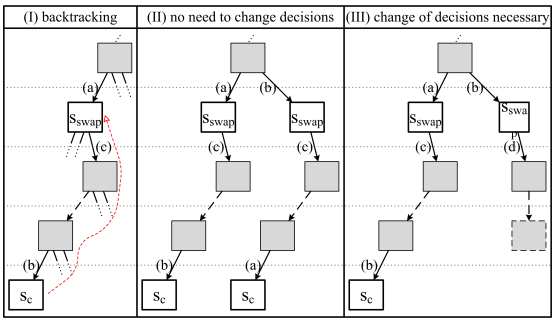

Fig. 1. Backtracking and consequences for the scheduling decisions

VI. Backtracking Heuristic

If the scheduler calculates a negative leeway for a slot, we start backtracking with our heuristic based on the leeway. In Figure 1 column (I), scheduling decision (b) for current slot s_c led to a negative leeway and hence, all succeeding scheduling decisions will yield infeasible schedule tables. As a consequence, we may skip the search for a feasible schedule in this part of the search tree. By this doing this, we save the time to check all succeeding decisions.

Based on the heuristic function (leeway), we look for a promising predecessor node to continue the scheduling process with a different scheduling decision for that node (see dotted arrow in Figure 1 column (I) from s_c to s_{swap}). We start in the current slot s_c and proceed upwards in the tree structure, checking based on the leeway for a slot s_{swap} at which we can swap the scheduling decision. The conditions for this swapping slot are: The swapping slot must be later than the release time of job i scheduled by decision (b). Furthermore, the leeway of a candidate for the swapping slot must be greater than or equal to the difference in number of slots between the current slot and the candidate for the swapping slot, i.e. $\delta(s_{swap}) \geq s_c - s_{swap}$. If these conditions are fulfilled, we can delay scheduling decision (a) for the swapping slot.

Once we have found a slot which fulfills the swapping conditions, we swap scheduling decisions (b) and (a), hence, decision (b) is now taken for s_{swap} and decision (a) for s_c (see Figure 1 column (II)). The scheduling decisions after the swapping slot – e.g. decision (c) – remain unchanged. After swapping of the decisions of the two slots, we have to recalculate the leeways of these slots.

By swapping scheduling decisions (a) and (b), it is possible that fulfilled precedence constraints are not fulfilled anymore and/or scheduled demand of HI criticality jobs J_i^{Δ} is changed. In this case, we cannot continue with scheduling decision (c)

after s_{swap} and we continue the scheduling process after s_{swap} with a possibly different decision (d) for the next slot (see Figure 1 column (III)). As a result, the current slot is set to the swapping slot and we continue the scheduling process from that slot.

Depending on the scheduled jobs in the current slot and the swapping slot, there are different cases which we have to consider when making swapping decisions. In the following, we present these cases and the consequences for the scheduling process. We refer to slots in mode LO-*table* by s^{LO} and slots in mode HI-*table* by s^{HI}.

Case 1: In both slots s_c^{LO} and s_{swap}^{LO}, a LO criticality job is scheduled. In s_{swap}^{HI}, no job is scheduled (Figure 2). We swap slots in LO criticality mode LO-*table* and update the leeway of s_c and s_{swap}. The swapping slot in the HI criticality mode remains unchanged. In a last step, we schedule the current slot in the HI criticality mode. Swapping of this case does not change any precedence constraints and this refers to Figure 1 column (II).

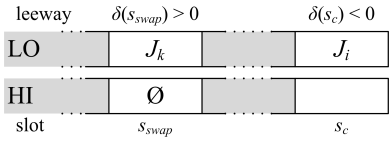

Fig. 2. Backtracking case 1: slots before backtracking

Case 2: In both slots s_c^{LO} and s_{swap}^{LO}, a LO criticality job is scheduled. In s_{swap}^{HI}, a HI criticality job J_m^{Δ} is scheduled (Figure 3). We swap slots in LO criticality mode LO-*table* and update the leeway of s_c and s_{swap}. Swapping two LO criticality jobs does not change precedence constraints, and hence, the scheduled job in s_{swap}^{HI} remains unchanged. In the last step, we schedule the current slot in the HI criticality mode. This case refers to Figure 1 column (II).

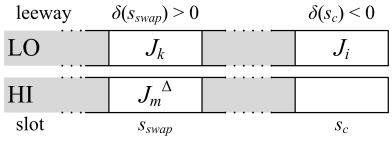

Fig. 3. Backtracking case 2: slots before backtracking

Case 3: In slot s_c^{LO}, a LO criticality job J_i and in slot s_{swap}^{LO}, a HI criticality job J_k^{LO} are scheduled. In s_{swap}^{HI}, the same HI criticality job J_k^{LO} as in s_{swap}^{LO} is scheduled (Figure 4).We swap slots in LO criticality mode LO-*table* and update the leeway of s_c and s_{swap}. Now, we must check whether fulfilled precedence constraints have been changed by swapping. If fulfilled precedence constraints have been changed then we re-schedule slot s_{swap}^{HI} and set the swapping slot as current slot and continue the scheduling process. This refers to Figure 1 column (III). If fulfilled precedence constraints have not been changed then we swap s_{swap}^{HI} and s_c^{HI} (unscheduled yet) and re-schedule s_{swap}^{HI} based on the fulfilled precedence constraints at that time – as described in subsection V-B. This refers to Figure 1 column (II).

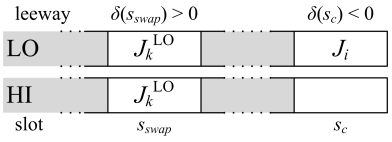

Fig. 4. Backtracking case 3: slots before backtracking

Case 4: In slot s_c^{LO}, a HI criticality job J_i^{LO} and in slot s_{swap}^{LO}, a LO criticality job J_k are scheduled. In s_{swap}^{HI}, no job is scheduled (Figure 5). We swap slots in LO criticality mode LO-*table*. In slot s_{swap}^{HI}, we scheduled the same job J_i^{LO} as in s_{swap}^{LO} (after swapping). As a consequence, we update $g_{sched}^{\Delta}(s)$ and leeway $\delta(s)$ for $s \in \{s_{swap}, .., s_c\}$. In the last step, we schedule the current slot in the HI criticality mode. This case refers to Figure 1 column (II).

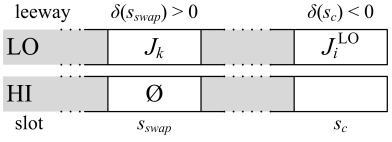

Fig. 5. Backtracking case 4: slots before backtracking

Case 5: In slot s_c^{LO}, a HI criticality job J_i^{LO} and in slot s_{swap}^{LO}, a LO criticality job J_k are scheduled. In s_{swap}^{HI}, a HI criticality job J_m^{Δ} is scheduled (Figure 6). First, we check whether scheduling J_m^{Δ} in s_c^{HI} leads to a deadline miss. If yes, then we have to search for another swapping slot. If no, then we swap slots in LO criticality mode LO-*table*. Then, we swap s_{swap}^{HI} and s_c^{HI} (unscheduled yet) and schedule in s_{swap}^{HI} the same job J_i^{LO} as in slot s_{swap}^{LO} (after swapping in LO-*table*). As a consequence, we update $g_{sched}^{\Delta}(s)$ and leeway $\delta(s)$ for $s \in \{s_{swap}, .., s_c\}$. This case refers to Figure 1 column (II).

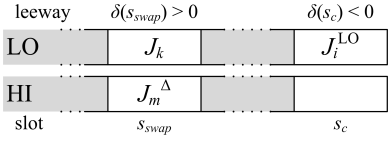

Fig. 6. Backtracking case 5: slots before backtracking

Case 6: In both slot s_c^{LO} and s_{swap}^{LO}, a HI criticality job J_i^{LO} and J_k^{LO} are scheduled. In s_{swap}^{HI}, the same HI criticality job J_k^{LO} is scheduled as in slot s_{swap}^{LO} (Figure 7). We swap slots in LO criticality modes and update the leeway of s_c and s_{swap}. Now, we must check whether fulfilled precedence constraints are not fulfilled after swapping anymore. If fulfilled precedence constraints have been changed, then we schedule J_i^{LO} in slot s_{swap}^{HI}, set the swapping slot as current slot, and continue the scheduling process. This refers to Figure 1 column (III). If fulfilled precedence constraints have not been changed, then we schedule J_i^{LO} in slot s_{swap}^{HI}. This refers to Figure 1 column (II).

VII. Evaluation

In this section, we show first evaluation results. We evaluated our algorithm by generating job sets with uniformly distributed utilizations by means of the UUniFast algorithm [8]. We generate a set of periodic tasks and unroll them into a set

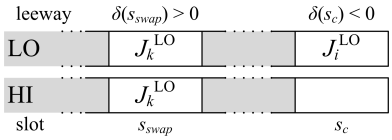

Fig. 7. Backtracking case 6: slots before backtracking

of jobs which are then handled by our algorithm. We use these synthetic workloads to show the correctness for general workloads. Rounding to full slot size values causes errors in the obtained utilization, hence, we specified that resulting job sets have errors less than 3%. We generated the mixed criticality job sets with the following constraints: the demand of all jobs with $C(\mathrm{LO})$ and the demand of HI criticality jobs with $C(\mathrm{HI})$ (original job parameters before splitting) have to be feasible by demand.

As input parameters, we used the utilization of all tasks with $C(\mathrm{LO})$ and the ratio of HI criticality jobs within each job set, which is in line with [7]. We evaluated the set of utilizations from 10% to 80% in steps of 10% with a ratio of HI criticality jobs of 50%. The range of LO criticality WCETs is $C_i(\mathrm{LO}) \in [1; 15]$. For the WCETs of HI criticality jobs, we used a high-scale factor $hsf = 3$, i.e. $C_i(\mathrm{HI}) \in [C_i(\mathrm{LO}); hsf \cdot C_i(\mathrm{LO})]$. The range of relative deadlines D_i was chosen within the interval $[45; 120]$ For each combination of input parameters, we generated $N = 1,000$ random job sets and scheduled them with our scheduler. We evaluated also other ratios of high criticality jobs and high scale factors, but for space reason, we only show a representative example.

Table II shows the success ratio of scheduling the generated job sets. The results are plotted in Figure 8. Due to $hsf = 3$,

success ratio	$\sum_{i=1}^{n} U_i(\mathrm{LO})$							
	10%	20%	30%	40%	50%	60%	70%	80%
swapping	100.0	100.0	100.0	90.9	14.6	1.1	0.2	0.0
FPS	100.0	100.0	99.6	70.2	3.5	0.0	0.0	0.0

TABLE II
RESULTS: SUCCESS RATIO FOR $N = 1000$ AND $hsf = 3$

it is possible to obtain a utilization for all jobs under CAs' assumption of nearly 100% for low criticality utilizations above 30%. This leads to a drop of the success ratio, whereas the fixed priority approach is affected more strongly.

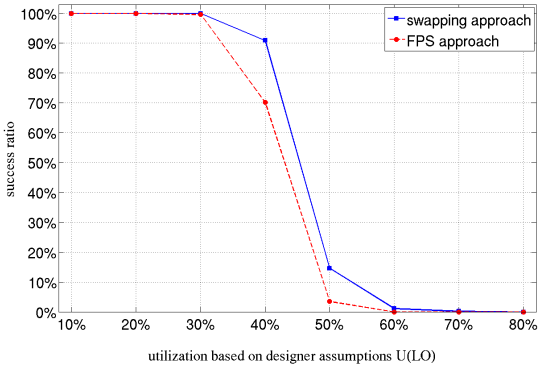

Fig. 8. Comparison between FPS approach [5] and our approach

VIII. CONCLUSION

Due to its run-time simplicity, extreme determinism, and ease of validation, the TT approach to real-time scheduling is heavily favored in industrial practice in many safety-critical application domains. In [5] an effort was made to extend results from the recently emergent field of mixed criticality scheduling to time-triggered scheduling, by proposing a TT-based framework for implementing mixed criticality systems, and presenting proof-of-concept algorithms for generating the schedule tables needed by this framework.

In this paper, we have extended and generalized the work described in [5]. We presented an algorithm for handling mixed criticality applications in time-triggered systems replacing these proof-of-concept methods with an algorithm for run-time execution and construction of schedule tables for efficiency and flexibility, providing realistic applicability. Our algorithm for the construction of the schedule tables is search-based; it is implemented as a tree search with backtracking. We devised two heuristics, one for the construction of the schedule tables and another for backtracking, based on the demand of HI criticality applications. These heuristics allow for a reduction of the search space and the time-complexity for scheduling decisions and backtracking; in addition to immediately yielding a constructive proof of the correctness of the schedule tables.

Due to the search tree based scheduling, the algorithm will be augmented to consider further constraints, in future. For example, extending the search not only to find a feasible schedule but also a schedule minimizing the preemptions.

REFERENCES

[1] *ARINC 653-1 Avionics application software standard interface*. 2003.
[2] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, 1991.
[3] N. C. Audsley. *Flexible Scheduling in Hard-Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1993.
[4] S. Baruah, V. Bonifaci, G. D'Angelo, Haohan Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 2012.
[5] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Real-Time Systems Symposium*, 2011.
[6] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems Journal*, 1990.
[7] S.K. Baruah, A. Burns, and R.I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS)*, 2011.
[8] Enrico Bini and GiorgioC. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30:129–154, 2005.
[9] A. Burns and R. Davis. Mixed criticality systems: a review. www-users.cs.york.ac.uk/ burns/review.pdf.
[10] D. de Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *RTSS*, 2009.
[11] G. Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE Transactions on Information and Systems*, 1993.
[12] G. Fohler. *Flexibillity in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, 1994.
[13] H. Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2011.
[14] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 1985.
[15] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *International Conference on Embedded Software*, 2011.