Time-Triggered Mixed-Critical Scheduler

Dario Socci, Peter Poplavko, Saddek Bensalem and Marius Bozga
UJF-Grenoble 1
CNRS VERIMAG UMR 5104,
Grenoble, F-38041, France
{*Dario.Socci* | *Petro.Poplavko* | *Saddek.Bensalem* | *Marius.Bozga*}*@imag.fr*

***Abstract*—Modern safety-critical systems, such as avionics, tend to be mixed-critical, because integration of different tasks with different assurance requirements can effectively reduce their costs. Scheduling is one of the main challenges of such systems. In this work we show that a generalization of the Time Triggered (TT) scheduling paradigm, Single Time Table per Mode (STTM) dominates other approaches like Fixed Priority or Fixed Priority per Mode (FPM). We also propose an algorithm to transform any FPM priority assignment to an equivalent set of STTM tables.**

I. Introduction

Advances in technology is leading towards an increasing trend in integration of multiple functionalities on a single chip. Integration is an effective way of reducing cost and power consumption of embedded systems. On the other hand, for safety-critical domains, such as avionics, this is leading to the integration of tasks with significantly *asymmetric* safety requirements on a single assembly of processing resources.

Such system have called into existence a special Mixed-Critical System (MCS) scheduling theory, that has been developed at least since 2007 [1]. This theory treats the asymmetric safety requirements by adequate scheduling methods, which leads to much more efficient resource usage compared to classical scheduling approaches [2]. In particular, MCS-aware scheduling methodologies were demonstrated in [3] to significantly outperform traditional approaches such as reservation-based techniques.

A major branch of the MCS scheduling theory is the *certification-cognizant* mixed-critical scheduling, which assumes that all tasks are hard real-time and supports the certification prescribed by safety-critical standards such as $DO-178B$ [4]. Although this approach follows these prescriptions in a rather simple pragmatic way, it faces NP-complete problems even under basic assumptions [3].

Following a significant volume of previous MC scheduling work (*e.g.,* [3], [5], [6], [7]) in this paper we consider the basic problem of single-core scheduling for a finite set of jobs whose exact arrival times are known *a priori*. As stated in [6], this assumption applies without restrictions when generating schedules for *time-triggered architecture*, in this case one can just apply a finite job algorithm, like the one presented in this paper, to a hyperperiod of periodic tasks. We also restrict ourselves to dual-critical problems. This restriction is often assumed in literature ([5], [6], [7], [8], [9]). Dual-critical system are also of practical interest, such applications as *Unmanned aerial vehicles* (UAVs) assume two criticality levels: *safety critical* and *mission critical*.

The Own-Criticality Based Priority (OCBP) [5] is theoretically the best among all *Fixed Priority per job* (FP) scheduling algorithms for MCS. Recent extensions of the fixed job priority policy [8], [9] perform a switch between different priority tables for different modes. This Fixed Priority per Mode per job (FPM) policy can lead to better results due to their higher flexibility. In particular Mixed-Critical Earliest Deadline First (MCEDF) [10] has been proven to dominate OCBP, and hence FP scheduling, for dual-critical problems.

In [6] Baruah *et al.* propose a Time Triggered (TT) version of OCBP. This scheduling algorithm uses one static table per criticality mode. We will call this approach Single Time Table per Mode (STTM). The Time Triggered algorithms are important because they are easy to certify, since the time intervals in which each job executes are statically known *a priori*, while in an approach like FPM the jobs can interact in different ways depending on the execution times. The only unknown variable in STTM scheduler is the time when a switch will occur, but there are only a small number[1] of precomputed instants of time where this can happen, while in FPM these are infinite. Thus, even if STTM is a dynamic scheduler, all the possible executions can be easily enumerated, making certification easier. Also some commercial systems implement TT as default scheduling mechanism. In general, it is NP-complete problem to decide whether optimal scheduling policy (OPT) exists.

This work focuses on STTM algorithms, giving two main contributions: we prove that STTM approach dominates FPM and we give an algorithm that allows to transform an FPM priority assignment into a set of STTM tables. The following gives a relation between the sets of schedulable instances for dual-critical problems:

$$FP \overset{1}{\subsetneq} MCEDF \overset{2}{\subsetneq} FPM \overset{3}{\subsetneq} STTM \subseteq OPT \qquad (1)$$

Inclusion 1 is proved by dominance of MCEDF over OCBP [10]. Next, Inclusion 2 is true by definition of MCEDF. We can easily prove that the inclusion is strict under the hypothesis that $P \neq NP$. In fact under the restrictive hypothesis that all arrival times are equal to zero, we have that FPM is optimal, but the problem remains NP-complete even under this assumption [3]. If we assume by contradiction that MCEDF=FPM, then MCEDF could solve NP-complete problems in polynomial time. Example A.1 shows an instance that is FPM-schedulable but not MCEDF-schedulable.

The research leading to these results has received funding from **CERTAINTY** – European Community's Seventh Framework Programme [FP7/2007-2013], grant agreement no. 288175.

[1] equal to the number of HI jobs in a dual-critical system

Inclusion 3 is the main contribution of this paper. We will prove in Section IV that there exists an algorithm that can transform any FPM priority assignment into STTM tables. The strictness of the inclusion is shown by Example IV.1. Note that since the FPM policy is completely defined by a finite set of *basic* scenarios [3], such scenarios could be used as tables for a TT-like scheduling. This is theoretically feasible, but it has little practical interest, since it would potentially require a huge number of tables, whereas we require only two tables. Hence in this paper we actually propose a TT extension for MCEDF or any other FPM policy.

II. Background

Consider a set of hard real-time jobs having different *levels of criticality*. It is common in literature to model different criticality requirements by giving different worst-case execution times (WCETs) for the same job. In *dual-criticality* systems we have the highly level, denoted as 'HI', and the low critical (normal) level, denoted as 'LO'. Every job gets a pair of WCET values: the LO WCET and the HI WCET. One important remark is that both HI and LO jobs are hard real-time, so both *must* terminate their executions before the deadlines. But only HI jobs undergo certification. This means that the designer is confident that the jobs will never exceed their LO WCET, calculated by exhaustive measurements and adding some practical margin. However, it is required to prove to the certification authorities that the HI jobs will meet the deadlines even under the unlikely event that some jobs would execute at their HI WCET, calculated by more safe and pessimistic formal WCET estimation tools, required for certification.

A. Formalism

In a dual-criticality MCS, a *job* J_j is characterized by a 5-tuple $J_j = (j, A_j, D_j, \chi_j, C_j)$, where:

- $j \in \mathbb{N}_+$ is a unique index
- $A_j \in \mathbb{Q}$ is the arrival time, $A_j \geq 0$
- $D_j \in \mathbb{Q}_+$ is the deadline, $D_j > A_j$
- $\chi_j \in \{\text{LO}, \text{HI}\}$ is job's criticality level
- $C_j \in \mathbb{Q}_+^2$ is a vector $(C_j(\text{LO}), C_j(\text{HI}))$ where $C_j(\chi)$ is the WCET at criticality level χ.

The index j is technically necessary to distinguish between the jobs with the same parameters. We assume that $C_j(\text{LO}) \leq C_j(\text{HI})$. We also assume that the LO-criticality jobs are forced to terminate after $C_j(\text{LO})$ time units of execution, so $(\chi_j = \text{LO}) \Rightarrow C_j(\text{LO}) = C_j(\text{HI})$. An *instance* $\mathbf{J}$ of the MC-scheduling problem is a set of K jobs. A *scenario* of an instance $\mathbf{J}$ is a vector of execution times of all jobs: $(c_1, c_2, \ldots, c_K)$. If at least one c_j exceeds $C_j(\text{HI})$, the scenario is called *erroneous*. The *criticality of scenario* $(c_1, c_2, \ldots, c_K)$ is the least critical χ such that $\forall j\ ,\ c_j \leq C_j(\chi)$. A scenario is *basic* if for each j either $c_j = C_j(\text{LO})$ or $c_j = C_j(\text{HI})$.

A (preemptive) schedule is a mapping from physical time to $\mathbf{J} \cup \{\perp\}$, where $\perp$ denotes no job. Every job should start at time A_j or later and run for no more than $C_j(\text{HI})$ time units. The online state of a run-time scheduler at every time instance consists of the set of terminated jobs, the set of *ready jobs*, *i.e.,* jobs that have arrived in the past and did not terminate yet, the progress of ready jobs, *i.e.,* how much each of them has executed so far, and the current criticality mode, χ_{mode}, initialized as $\chi_{mode} = \text{LO}$ and switched to 'HI' as soon as a HI job exceeds $C_j(\text{LO})$. A schedule is *feasible* if the following conditions are met:

Condition 1. *If all jobs run at most for their LO WCET, then both critical (HI) and non-critical (LO) jobs must terminate before their deadline.*

Condition 2. *If at least one job runs for more then its LO WCET, than all critical (HI) jobs must terminate before their deadline, whereas non-critical (LO) jobs may be even dropped.*

An instance $\mathbf{J}$ is *clairvoyantly schedulable* if for each non-erroneous scenario, when it is known in advance (hence *clairvoyantly*), one can specify a feasible schedule. By default, the scheduling is non-clairvoyant.

Based on the online state, a *scheduling policy* deterministically decides which ready job is scheduled at every time instant. A scheduling policy is *optimal* (or *correct*) for the given instance $\mathbf{J}$ if for each non-erroneous scenario it generates a feasible schedule. We assume without loss of generality that the scheduling policies are *monotonic per scenario*, which means one can check their optimality by simulating for all basic scenarios [3]. A *mode-switched* scheduling policy uses χ_{mode} in the scheduling decisions, *e.g.,* to drop the LO jobs, otherwise it is *mode-ignorant*. An instance $\mathbf{J}$ is *MC-schedulable* if there exists an optimal scheduling policy for it. A *fixed-priority* scheduling policy is a mode-ignorant monotonic policy that can be defined by a priority table PT, which is a K-sized vector specifying all jobs (or, optionally, their indexes) in a certain order. The position of a job in PT is its *priority*, the earlier a job is to occur in PT the higher the priority it has. Among all ready jobs, the fixed-priority scheduling policy always selects the highest-priority job in PT. If a scheduling policy cannot be defined by a static priority table, it is called *dynamic-priority*.

A Time-Triggered (TT) table is a static, pre-computed table that defines at every instant of time which job must be scheduled. We define a Single Time Table per Mode (STTM) scheduling as an extension of TT scheduling that associates to each mode a single TT table.

In this paper, we consider the construction of STTM table starting from *fixed priority per mode* (FPM) policy, that is a fixed-priority scheduling that has a different PT for each mode. We assume that the scheduler is preemptive. For the given job set $\mathbf{J}$, this policy assumes two fixed-priority tables PT_{LO} before the mode switch and PT_{HI} after the mode switch. We assume that in the HI mode the LO jobs are dropped and hence excluded from PT_{HI}[2].

The basic scenario **LO** is the scenario where all jobs execute for time $C(\text{LO})$. Under the FPM policy, the basic scenario **HI-***J* is the scenario where job J is the first job that switches into HI mode after having executed for time $C(\text{LO})$.

[2]This assumption is legal according to Condition 2 and can only improve the schedulability

After the switch, job J and all non terminated HI jobs execute for time $C(\text{HI})$.

If we simulate the FPM policy for **LO** and all **HI-*J*** scenarios, we obtain function E^{LO} and E^{HI-J}, defined as $E^{LO|HI-J} : Time \rightarrow \{\perp\} \cup \mathbf{J}$ that specify for every time instance the job that runs at given time instance or $\perp$ when the processor is idle, when one job preempts another, or when a job starts/terminates. Note that this leads to *open* intervals of job activity and closed or single-point intervals of idle processor. These functions could be used for time-triggered scheduling, where we start in **LO** table defined by E^{LO} and switch to **HI-*J*** table defined by E^{HI-J} whenever a given HI job J switches to HI mode. But this would require an individual table per HI job, which is of little practical value.

III. Transformation Algorithm

Our goal is to obtain one single table **HI*** for the switch to the HI mode by *any* HI job. In this section we will show how to build such a table, while in Section IV we will show its correctness. We propose a method to generate this table by simulating fixed-priority for HI jobs with $C(\text{HI})$ times using priority table PT_{HI} and assuming that a HI job can be disabled at any time when all three *enabling* rules defined below are false. Whenever a non-terminated HI job is (temporarily) disabled, a lower-priority HI job can execute (priority inversion).

Before we give the rules, let us give some supplementary definitions.

Let $T_j^{LO}(t)$ (resp. $T_j^{HI*}(t)$) be the cumulative execution progress of job J_j by time t in table **LO** (resp. **HI***). We call a HI job that has executed for more than its $C(\text{LO})$ a *switched job*. We say that such a job switches at time t if $T_j^{LO}(t) = C_{LO}$. It is *non-switched* otherwise.

The method to generate **HI*** is as follows:

- at any time t, we execute the highest priority (according to PT_{HI}) *enabled* HI job
- a job J_j is *enabled* at time t if:
  - the job has arrived: $t > A_j$
  - the job has not yet executed for its HI WCET: $T_j^{HI*}(t) < C_j(\text{HI})$
  - at least one of the following *rules* is true:

$$T_j^{LO}(t) = C_j(\text{LO}) \quad (2a)$$
$$T_j^{HI*}(t) < T_j^{LO}(t) \quad (2b)$$
$$T_j^{HI*}(t) = T_j^{LO}(t) \wedge E^{LO}(t) = j \quad (2c)$$

Informally, Rule (2a) allows switched jobs to run as soon as possible, while Rules (2b) and (2c) assure that a job will not run in **HI*** for more time than in **LO** before the switch.

Example III.1. *Let us consider the following instance as an example:*

Job	*A*	*D*	χ	*C(LO)*	*C(HI)*
1	*0*	*12*	*HI*	*3*	*5*
2	*6*	*11*	*HI*	*2*	*4*
3	*7*	*8*	*LO*	*1*	*1*
4	*1*	*4*	*HI*	*1*	*2*

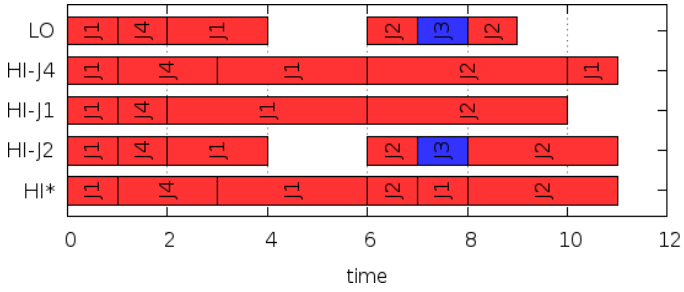

Fig. 1. Basic scenarios and TT tables

and assume the following FPM priority assignment[3]*:*

$$PT_{LO} = \quad J_4 \prec J_1 \prec J_2 \prec J_3$$
$$PT_{HI} = \quad J_4 \prec J_1 \prec J_2$$

*Figure 1 presents all the basic scenarios and the table **HI***. Consider this table. At time 0, only J_1 has arrived, and it is enabled by Rule (2c). At time 1, J_4 arrives, it has higher priority then J_1 and it is enabled by Rule (2c), so it is chosen by the algorithm to be executed. At time 2 for job J_4 Rule (2c) will be false, but Rule (2a) will become true, so we will continue execute it until time 3. At time 3 J_4 will terminate, so J_1 will be enabled by Rule (2b) until time 5 and by Rule (2a) from 5 on. So J_1 will continue its execution till time 6, when J_2 arrives. J_2 is enabled by Rule (2c), and it has higher priority than J_1, so it will be executed until time 7. At this instant Rule (2c) becomes false for J_2, disabling it. So we execute J_1. At time 8 J_1 terminates and J_2 is enabled by Rule (2c). At time 9 Rule (2c) is false for J_2, while Rule (2a) becomes true. So J_2 continues its execution until time 11, when it terminates.*

*It is easy to verify the correctness of TT scheduling that uses **LO** and **HI*** as tables. In fact in table **LO** all the jobs meet the deadline. When there is a switch, at time t, from **LO** to **HI***, all HI job J_j must have from time t a quantity of time reserved for them in **HI*** equal to $C_j(HI) - T_j^{LO}(t)$. In our example, if there is a switch in the **LO** table at time 2, caused by job J_4, then J_1, J_4 and J_2 will have enough remaining time reserved in **HI*** (respectively $4 = C_1(HI) - T_1^{LO}(2) = 5 - 1$, $1 = C_4(HI) - 1$ and $4 = C_2(HI) - 0$), and will terminate before their deadlines. In this case we will drop job J_3, since we do not care about LO jobs when in HI mode. Similarly, in the case of a switch at time 4, caused by J_1, then J_1 and J_2 will have respectively $3 = C_1(HI) - 2$ and $4 = C_2(HI) - 0$. Note that in this case J_1 will have one time unit more then it actually needs. Finally, if there will be a switch at time 9, caused by job J_2, this job will have 2 other time units, terminating at time 11, meeting its deadline.*

IV. Proof of Dominance

We will prove in this section the Inclusion 3 of Equation (1). At the same time we also provide an algorithm to construct STTM tables from FPM priority tables, generated by an algorithm such as MCEDF.

The following is the main claim of this paper:

[3]Can be computed using MCEDF [10]

Theorem IV.1. *If the FPM policy leads to a feasible schedule, then a switched time triggered schedule that uses **LO** and **HI*** as, respectively, LO-mode and HI-mode table, is a feasible schedule as well.*

To prove it, we first show that:

Lemma IV.2. *If at any time we switch from **LO** to **HI***, then all the unterminated jobs will have enough time reserved in **HI*** to terminate their work.*

First, let us comment that, according to our rules to construct **HI***, no HI jobs get disabled forever because eventually Rule (2a) becomes true, since all LO jobs eventually terminate. Thus, all HI jobs get a total time $C(\text{HI})$ reserved in **HI***. Consequently, if a job switches at time t, then this and any other job is guaranteed to get $C(\text{HI}) - T_j^{HI*}(t)$, but needs to get at least $C(\text{HI}) - T_j^{LO}(t)$.

Therefore the lemma can be equivalently stated as follows:

*no non-switched HI job makes more progress in **HI*** than in **LO**.*

Formally:

$$\forall t \ , \ T_j^{LO}(t) < C_j(\text{LO}) \Rightarrow T_j^{LO}(t) \geq T_j^{HI*}(t)$$

Proof of Lemma IV.2: At time $t = 0$ the lemma thesis is obviously true, and with progress of time it can be invalidated only during the time when a job is scheduled in **HI***. However, as long as $T_j^{LO}(t) < C_j(\text{LO})$ job J_j can only be scheduled when either (2b) or (2c) is true, but they both imply that we have $T_j^{LO}(t) \geq T_j^{HI*}(t)$. ■

Let $TT_J^{HI(LO|HI-J')}$ be the termination time of J in **HI*** (respectively, **LO**, **HI-***J***'**).

Theorem IV.3. *Let J^{least} be the least priority job in PT_{HI}, then $\exists J' \ : \ TT_{J^{least}}^{HI*} \leq TT_{J^{least}}^{HI-J'}$*

Let us first give some definitions and support lemmas. A *busy interval* in some table (be it **LO**, **HI-***J* or **HI*** table) is a maximal continuous interval of time where some jobs are enabled for execution, where for table **HI*** we apply special rules defined earlier which can disable a job temporarily. When such rules are not applied, the busy intervals are obviously *open* intervals, because they are composed of union of (intersecting) open intervals between arrival and termination of different jobs. We state without proof that even with the extra rules we defined earlier for **HI***, the busy intervals remain to be open intervals.

For convenience, we use the term 'busy interval' also for the set of jobs that are enabled at least once inside the busy interval, and denote it BI, *e.g.,* BI^{HI*} for busy intervals in **HI***. Note that for this table, unlike the other tables, it is not always so that the total interval duration is exactly equal to the total work of jobs in BI, because there are rules that can temporarily disable a job after its arrival and before its termination. Therefore, the total work of jobs in BI^{HI*} can exceed the length of the busy interval. This also means that a job may belong to several busy intervals of **HI***.

In between BI, there are closed, sometimes single-point, *idle intervals*. For **HI***, we would like to distinguish an idle interval as a *hole* if inside this interval there are HI jobs that have arrived and not yet terminated, and are disabled because neither of the rules (2a), (2b), (2c) is true. The idle intervals that are not holes, are called *empty* intervals, *i.e.,* those where the job queue is empty.

For instance in Figure 1 in **HI*** there are two busy intervals: (0,8) and (8,11), thus we have a hole of size 0 at time 8. This happens because we have that immediately before time 8 J_1 is enabled by Rule (2a) while J_2 is disabled. On the other hand, at time 8 J_1 is disabled (because it terminates) while J_2 is enabled by Rule (2c).

The following proposition is well-known for fixed-priority policies, but needs to be re-established because we added the rules that can disable jobs.

Lemma IV.4. *If J^{least} is the least priority job in PT_{HI} then it terminates at the end of some busy interval BI^{HI*}.*

Proof: Let us assume by contradiction that J^{least} terminates inside a busy interval at time t. This means that at time t there is another enabled job (by definition of busy interval). If that is so, then J^{least}, having the least priority, should not be running at time t. ■

Lemma IV.5. *Let $BI^{HI*} = (a, b)$ be a busy interval in **HI***. At time a, the set of non-terminated HI jobs is the same in tables **LO** and **HI***, and for all of them holds that at time a the cumulative execution progress in **LO** is the same as in **HI***.*

Proof: Consider time a. The lemma thesis is obvious for any job that did not arrive yet, so in the sequel we consider only those jobs that have arrived.

If a job J is non-terminated in **LO** then it is non-terminated in **HI*** as well by Lemma IV.2. In addition, by the same lemma we have:
(I) $T_J^{HI*}(a) \leq T_J^{LO}(a)$.

On the other hand, if job J is non-terminated in **HI*** then the fact that it is not enabled at time a (by lemma condition) implies that Rule (2a) is false and hence the job is non-terminated in **LO** as well. Combined with the earlier observations, we conclude that the sets of non-terminated jobs at time a in these two tables are equal. In addition, also Rule (2b) is false, which means:
(II) $T_J^{HI*}(a) \geq T_J^{LO}(a)$.
Combining (I) and (II) we have the equality of the cumulative times. ■

Corollary IV.6. *Let $BI^{HI*} = (a, b)$ be a busy interval in which some job switches. Let J_s be the first such job, and let t^s be the time at which the switch occurs.*

*Then during the interval (a, t^s) tables **HI***, **HI-**J_s and **LO** are identical*

Proof: Notice that **HI-**J_s and **LO** are equal by construction in $(0, t^s)$ and hence in (a, t^s) as well. Let us compare **LO** and **HI***. At time a the set of non terminated jobs in these two tables are equal. In interval (a, t^s) no job switched yet, therefore all the jobs that run in **HI*** should satisfy Rule (2c), which is due to the fact that the other two rules require a switch

to have occurred. As long as Rule (2c) holds, the **HI*** table replicates the **LO** table, and because it fills time interval (a, t_s) continuously, as $t_s \in BI^{HI*}$, we have proved our thesis.

■

Proof of Theorem IV.3: Let $BI^{HI*} = (a, b)$ be the busy interval in which J^{least} terminates. By Lemma IV.4, $TT^{HI*}_{J^{least}} = b$. By Lemma IV.5, job J^{least} is not yet switched at start of this interval, and since this job terminates at the end of BI^{HI*}, we know also that it switches inside this interval as well, so Corollary IV.6 applies for this interval.

Let us assume that $BI^{HI*} = (a, b)$ is followed by an empty interval, *i.e.,* an idle interval which appears due to termination of all HI jobs that have arrived so far. Because in this case all the jobs of BI^{HI*} have terminated by time b, we have:

$$b = a + \sum_{j \in BI^{HI*}} \left(C_j(\text{HI}) - T_j^{HI*}(a)\right)$$

Let J_s be the first job to switch in BI^{HI*}, at time t^s. By Lemma IV.5 and Corollary IV.6, we have that the same jobs, with the same remaining execution time as in **HI*** will run from time a in **HI-**J_s before the switch and, by construction after the switch as well. Therefore $BI^{HI*} = BI^{HI-J_s}$ and J^{least}, being the least-priority job, will terminate at time b in both tables.

Let us now examine the other case, in which $BI^{HI*} = (a, b)$, the busy interval where J^{least} terminates, is followed by a hole, *i.e.,* the idle interval which appears because at time b the rules for table **HI*** have disabled the non-terminated jobs. Also in this case J^{least} by our hypothesis and Lemma IV.4 will terminate at time b, but in this case by construction not all jobs of BI^{HI*} terminate by time b:

$$b < a + \sum_{j \in BI^{HI*}} \left(C_j(\text{HI}) - T_j^{HI*}(a)\right) \tag{3}$$

Let J_s be the first job to switch in BI^{HI*}, at time t^s. Again by Lemma IV.5 and Corollary IV.6 we observe the same initial state and subsequent behavior in tables **HI*** and **HI-**J_s of all non-terminated HI jobs during the time interval $(a, t^s]$. So we conclude that all jobs of BI^{HI*} run in **HI-**J_s after time a continuously, at time a their total remaining work is equal to:

$$\sum_{j \in BI^{HI*}} \left(C_j(\text{HI}) - T_j^{HI*}(a)\right)$$

In line with equation (3), in order to complete this workload, table **HI-**J_s has to continue execution after time b. New jobs may arrive before the termination of the busy interval BI^{HI-J_s}. this busy interval executes all these jobs, J^{least} being the last one to terminate. So we have:

$$BI^{HI*} \subseteq BI^{HI-J_s}$$

and

$$TT^{HI-J_s}_{J^{least}} \geq a + \sum_{j \in BI^{HI*}} \left(C_j(\text{HI}) - T_j^{HI*}(a)\right) \tag{4}$$

Combining (3) and (4), and observing that $TT^{HI*}_{J^{least}} = b$, we have that also in this case in **HI-**J_s the least-priority job terminates no earlier than in **HI***. This completes the proof of Theorem IV.3. ■

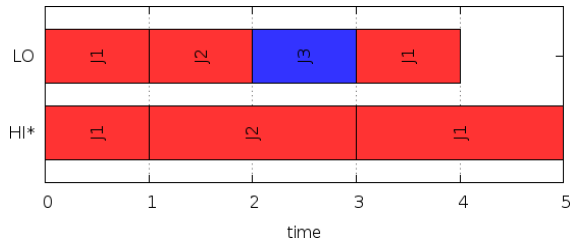

Fig. 2. The TT tables for the instance of Example IV.1

Proof of Theorem IV.1: From Lemma IV.2 we know that in any possible scenario all the HI jobs will have enough processor resource to terminate. The termination time of J^{least} is guaranteed to meet the deadline due to the hypothesis that it meets deadline in the FPM policy and Theorem IV.3. Now let us prove that also the HI jobs with higher priority in PT_{HI} meet their deadlines. Let $J^{\overline{least}}$ be the next least priority HI job after J^{least} in the PT_{HI} table. Let **J** be the currently examined problem instance and let $\overline{\mathbf{J}}$ be the instance obtained from **J** by reducing the criticality of J^{least} to LO. It is easy to show that the HI-mode table $\overline{\mathbf{HI}}*$ obtained for this new instance coincides with **HI*** except that the intervals where J^{least} is running are idled. So, $J^{\overline{least}}$ will terminate in **HI*** at the same time as in $\overline{\mathbf{HI}}*$, where by Theorem IV.3 applied to instance $\overline{\mathbf{J}}$ it will terminate no later than the latest termination under FPM policy. Obviously, also the latest termination of the FPM policy for job $J^{\overline{least}}$ is the same for both **J** and $\overline{\mathbf{J}}$. Because by our hypothesis this policy is feasible we conclude that $J^{\overline{least}}$ meets its deadline. Iterating this reasoning recursively, we argue that all HI jobs meet their deadline in **HI***, and thus we have our thesis. ■

Theorem IV.1 proves that FPM $\subseteq$ STTM. To prove the strictness of the inclusion, we give the following counter-example:

Example IV.1. *Let* $\mathbf{J}_d$ *be the following instance:*

Job	*A*	*D*	χ	$C(LO)$	$C(HI)$
1	*0*	*5*	*HI*	*2*	*3*
2	*1*	*3*	*HI*	*1*	*2*
3	*0*	*3*	*LO*	*1*	*1*

No FPM policy would schedule it. The only correct scheduling policy for $\mathbf{J}_d$ *is to execute* J_1 *for 1 time unit, then* J_2*. If* J_2 *terminates after 1 time unit, we execute* J_3 *and then* J_1 *again, otherwise we drop* J_3 *and execute* J_1*. It is easy to see that this is not an FPM schedule, as* J_1 *changes its priority* w.r.t. J_3 *in the LO scenario. This scheduling policy can be implemented using STTM tables as shown in Figure 2*

V. Conclusions and Future Work

In this paper we proposed a method to transform any FPM priority assignment into a set of STTM tables. This has a practical importance since safety critical systems are designed with a TT (also known as *static*) scheduler. A TT scheduler has a behavior that is completely precomputed. This makes certification of such system much easier. Although STTM is not static, it has a finite number of switches that can be trivially

checked the same way as TT. From a theoretical point of view, we proved that STTM dominates FPM.

In future work we plan to extend this algorithm for more than two levels of criticality. Also, it is necessary to investigate the mixed-critical scheduling of task graphs, where there are data dependencies between jobs.

REFERENCES

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium*, RTSS'07, pp. 239–243, IEEE, 2007.

[2] D. d. Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Real-Time Systems Symposium*, RTSS'09, pp. 291–300, IEEE, 2009.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Trans. Comput.*, vol. 61, pp. 1140 –1152, aug. 2012.

[4] L. A. Johnson, "DO-178B: Software considerations in airborne systems and equipment certification.," in *Radio Technical Commission for Aeronautics.*, RTCA, 1992.

[5] S. K. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium*, RTAS'10, pp. 13–22, IEEE, 2010.

[6] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pp. 3–12, 2011.

[7] T. Park and S. Kim, "Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems," in *Intern. Conf. on Embedded software*, EMSOFT '11, pp. 253–262, ACM, 2011.

[8] 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 *Euromicro Conf. on Real-Time Systems*, ECRTS'12, pp. 145–154, IEEE, 2012.

[9] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Euromicro Conf. on Real-Time Systems*, ECRTS'12, pp. 145–154, IEEE, 2012.

[10] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga, "Mixed critical earliest deadline first," in *Euromicro Conf. on Real-Time Systems*, ECRTS'13, pp. 93–102, IEEE, 2013.

APPENDIX

The following example proves that $MCEDF \subsetneq FPM$.

Example A.1. *Consider the following instance:*

Job	*A*	*D*	χ	$C(LO)$	$C(HI)$
1	*0*	*8*	*LO*	*5*	*5*
2	*0*	*10*	*HI*	*2*	*3*
3	*0*	*11*	*HI*	*2*	*5*

applying MCEDF[4] *to it we will have the following FPM priority assignment:*

$$\begin{aligned} PT_{LO} = &\quad J_2 \prec J_1 \prec J_3 \\ PT_{HI} = &\quad J_2 \prec J_3 \end{aligned}$$

It is easy to show that this priority assignment is not correct. In fact, if J_2 executes for $C_2(LO)$ and J_3 executes for $C_3(HI)$, then J_3 will terminate at time 12, missing its deadline.

[4]refer to [10] to know how to compute these priorities

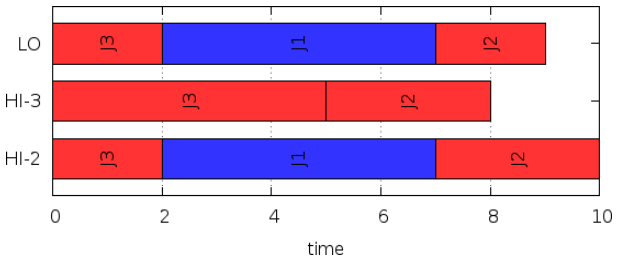

Fig. 3. Basic scenarios for Example A.1

On the other hand, the following FPM assignment:

$$\begin{aligned} PT_{LO} = &\quad J_3 \prec J_1 \prec J_2 \\ PT_{HI} = &\quad J_3 \prec J_2 \end{aligned}$$

is correct. This can be checked on the charts of Figure 3, where the basic scenarios are reported.