J Sched
DOI 10.1007/s10951-016-0468-y

An analysis of the non-preemptive mixed-criticality match-up scheduling problem

Zdeněk Hanzálek[1,2] · **Tomáš Tunys**[1] · **Přemysl Šůcha**[1]

Abstract Many applications have a mixed-criticality nature. They contain tasks with a different criticality, meaning that a task with a lower criticality can be skipped if a task with a higher criticality needs more time to be executed. This paper deals with a mixed-criticality scheduling problem where each task has a criticality given by a positive integer number. The exact processing time of the task is not known. Instead, we use different upper bounds of the processing time for different criticality levels of the schedule. A schedule with different criticality levels is generated off-line, but its on-line execution switches among the criticality levels depending on the actual values of the processing times. The advantage is that after the transient prolongation of a higher criticality task, the system is able to match up with the schedule on a lower criticality level. While using this model, we achieve significant schedule efficiency (assuming that the prolongation of the higher criticality task rarely occurs), and at the same time, we are able to grant a sufficient amount of time to higher criticality tasks (in such cases, some of the lower criticality tasks may be skipped). This paper shows a motivation for the non-preemptive mixed-criticality match-up scheduling problem arising from the area of the communication protocols. Using a polynomial reduction from the 3-partition problem, we prove the problem to be $\mathcal{NP}$-hard in the strong sense even when the release dates and deadlines are dropped and only two criticality levels are considered.

Keywords Uncertain processing times · Scheduling · Complexity · Robustness · Mixed-criticality · ILP · Match-up · Packing

1 Introduction

The problem we address in this paper is inspired by today's increasing trend to study *mixed-criticality* scheduling, which follows right from the attempts to provide multiple functionalities upon a shared resource (e.g., a machine performing production tasks or an embedded computer in a car performing computation and communication tasks). Because of that, the systems become mixtures of critical functionalities, which have to be executed, and non-critical functionalities, which may be skipped in situations when the critical ones need more processing time. Therefore, many applications are of mixed-criticality, where high-criticality tasks have to co-exist with low-criticality ones. They often use resources whose capacity is limited, and efficiency is a very important issue. At the same time, they are part of safety-critical systems (e.g., a brake-by-wire system in a car, or a just-in-time production for an important customer), where guaranteeing the integrity of the deadlines is of major importance. Therefore, we are faced with an interesting scheduling prob-

This work was supported by the US Department of the Navy Grants N62909-12-1-7009 and N62909-15-1-N094 issued by Office Naval Research Global. The United States Government has a royalty-free license throughout the world in all copyrightable material contained herein.

✉ Zdeněk Hanzálek
hanzalek@fel.cvut.cz

Tomáš Tunys
tunystom@fel.cvut.cz

Přemysl Šůcha
suchap@fel.cvut.cz

[1] Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic

[2] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic

Published online: 07 March 2016

lem occurring in the real-time systems (Sojka et al. 2011) and communication protocols (Hanzalek and Pacha 1998), since the guarantee of deadline (leading to sparse schedules when using pessimistic upper bounds of the processing/communication times) is often in contrast with efficiency (requiring dense schedules). Industry-relevant examples of challenging problems have been reported in Barhorst et al. (2009) and Lemke et al. (2012).

1.1 Motivation

This paper is motivated by safety-critical applications using the so-called time-triggered communication networks, where the nodes have synchronized clocks, and messages are transmitted at moments defined by the static schedule. Reliability of the messages used for critical functionalities of the system is typically increased by the message retransmission in conventional, event-triggered networks. The idea of this paper is to allow the retransmission of the safety-critical messages in the time-triggered networks as well. Therefore, the communication time of the safety-critical message depends on the number of retransmissions, which is observed at run time, i.e., it is not known a priori. Considering a highly used communication channel, the processing time prolongation needs to be compensated by skipping some of the non-critical messages.

Therefore, we formulate a uniprocessor scheduling problem (we consider the bus topology of the network, which is represented by one resource, i.e., one processor) with non-preemptive tasks (the task represents the message that has a given structure, and it cannot be preempted), and each task has its criticality (a processing time of the task on a given criticality level represents the communication time of the message including its retransmissions allowed for this criticality level). In this section, we will illustrate the problem on a case with two criticality levels (Low and High):

- the Low criticality level of the schedule (Lo-schedule) considers an optimistic upper bound of the processing times of both the Low criticality tasks (Lo-tasks) and High criticality tasks (Hi-tasks),
- the High criticality level of the schedule (Hi-schedule) considers a pessimistic upper bound of the processing times of the Hi-tasks (accounting for the message retransmission) and skipping the Lo-tasks.

The Lo-schedule and the Hi-schedule have to be synchronized, so that the system can switch at run time:

(1) from the Lo-schedule to the Hi-schedule, as a consequence of the Hi-task prolongation, which is observed at the completion time of the Lo-task (e.g., retransmissions of the safety-critical message) and

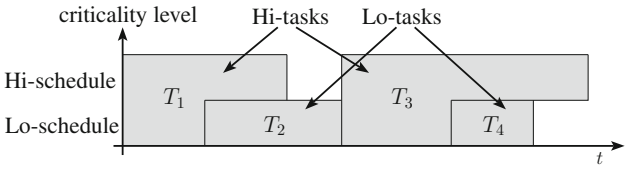

Fig. 1 Mixed-criticality schedule, where each row represents one criticality level and the Hi-tasks are represented by F shapes (the task is represented by a pattern in the form of the letter *F*)

(2) from the Hi-schedule to the Lo-schedule, to match up with the original schedule (e.g., to finish the retransmission before the start time of the subsequent Hi-task).

In fact, it is sufficient to synchronize both schedules at the start times of the Hi-tasks only. It is sufficient for case 1, because we deal with non-preemptive scheduling, and the optimistic upper bound of the processing time is shorter than the pessimistic one. It is also sufficient for case 2, because Lo-tasks do not exist in the Hi-schedule (i.e., Lo-tasks do not need to be synchronized). Illustration of the mixed-criticality schedule with four tasks on a uniprocessor is shown Fig. 1. If there are no time constraints, the $C_{\max}$ criterion (i.e., the schedule length given by the maximum completion time) can be improved while replacing task T_2 with task T_4.

This behavior may be seen as a variant of *match-up scheduling* (Aktürk et al. 2010). Given the task prolongation, match-up scheduling aims at finding a new schedule, which matches up with the original schedule at some point in the future, called the match-up time. In our case, the match-up time is equal to the start time of subsequent Hi-task (in order to ensure transmission of all safety-critical messages).

1.2 Related work

Existing research on mixed-criticality scheduling is aimed toward the construction of scheduling policies such that, on the one hand, they will not only help these mixed-criticality systems to facilitate the certification of critical functionalities, but, on the other hand, they will also make good use of the shared resources. Therefore, the literature studying mixed-criticality scheduling for embedded systems (Baruah et al. 2010) does not deal with two synchronized schedules, but it assumes only one switch from the Lo-schedule (providing maximum performance) to the Hi-schedule (satisfying the certification authority). Our objective is to obtain a schedule, which enables the match-up in order to provide a good performance after transient prolongation of a Hi-task.

It is well known that determining exact processing times of tasks is a very difficult problem (Nelson et al. 2015); instead, system engineers work with upper bounds on the exact value. However, for many non-trivial tasks, the best upper bounds that can be obtained are extremely pessimistic. Thus, the

strict constraints imposed by verification/certification of the safety-critical functionalities can cause very low resource use. Based on the observation that "the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice," Vestal (2007) proposed that different processing time values should be specified for each criticality level of a task, and we adopt this approach as well. Recent research in real-time systems has yielded some promising techniques for meeting the two aspects (timing properties and efficiency). This research has traditionally centered on an event-triggered approach to scheduling, see Burns et al. (2013).

The research of the mixed-criticality scheduling is, therefore, aimed toward the construction of scheduling policies so that, on the one hand, they will not only help these mixed-criticality systems to facilitate the certification process (see Baruah and Fohler 2011 on certification-cognizant time-triggered scheduling), but, on the other hand, they will also make good use of the shared resources. Following this idea, Theis et al. (2013) formulated the problem as a mode switch while constructing two schedule tables, one for the Lo and one for the Hi criticality mode so that switching from the Lo-table to the Hi-table is possible at every point in time. In Theis and Fohler (2013), Theis and Fohler presented an approach for the addition of mixed-criticality scheduling to legacy systems. It leaves the existing schedule table unchanged, only it provides analysis and adds a simple on-line mechanism to handle the changed criticality. According to the problem statement, these papers (Baruah and Fohler 2011; Theis et al. 2013; Theis and Fohler 2013) deal with time-triggered preemptive scheduling (the schedule table is divided into slots and tasks can be preempted at the slot borders) and switching back to the Lo criticality mode is not specified.

The basic model of *mixed-criticality* (MC) systems was formulated by Baruah, Li, and Stougie in Baruah et al. (2010), and in our paper we build on the terminology of this model. The problem of deciding whether the given MC instance with release dates and deadlines is schedulable or not (*MC Schedulability problem*) is $\mathcal{NP}$-hard in the strong sense (Baruah et al. 2012). In Baruah et al. (2010), the need for a new scheduling theory is justified by demonstrating the failure of the existing approaches (*reservations-based* and *priority-based*) in solving the MC Schedulability problem. These problems typically assume a preemptive execution of tasks in an event-triggered system. The above-mentioned works consider the execution level to be a non-decreasing function of time (i.e., the system is not expected to match up with the original schedule).

In the area of manufacturing, the problem of match-up scheduling (switching back to the original schedule after task prolongation) is related to the problems with controllable processing times (Aktürk et al. 2010; Shabtay and Steiner 2007) and problems with task rejection (Shabtay et al. 2013).

In the broader context we deal with the problem of robust optimization, which ensures that a chosen solution remains feasible and near optimal when uncertain parameters change somewhat (Bertsimas et al. 2011). The approach proposed in Soyster (1973) considers robust linear programming in which the columns of the coefficient matrix are uncertain and belong to a convex uncertainty set. Briand et al. (2007) considered a robust method, which characterizes a large set of optimal solutions allowing to switch from one solution to another, without any performance loss, in order to face potential disruptions which occur during the schedule execution.

Originally, worst-case scenarios were included in the uncertainty set, but this is typically too conservative for practical implementations. Detailed reviews of robust optimization with uncertainty sets can be found in Ben-Tal et al. (2009) and Bertsimas et al. (2011).

Resource and deadline constrained scheduling problems are typically formulated while incorporating both the decision and optimization part (for example, for the given input data, the problem is to find a schedule minimizing $C_{\max}$ or decide that no feasible schedule exists). The skipping (or rejection) of some deadline constrained tasks goes beyond the traditional definition of the scheduling problem, but it allows one to formulate the problem so that there is always some feasible solution (for example, the problem is to find a schedule minimizing the number of skipped tasks). Fung (2014) considered on-line preemptive scheduling problems where tasks have deadlines and the objective is to maximize the total weight of tasks completed before their deadlines. In Thevenin et al. (2014), the authors consider a single-machine scheduling problem with release dates, deadlines, sequence-dependent setup times, and a regular (i.e., non-decreasing) cost function associated with each task. Some of the tasks may be rejected, in which case a penalty is incurred. The field of scheduling with rejection provides schemes for coordinated sales and production decisions by grouping them into a single model. As explained in Slotnick's review paper (Slotnick 2011), the problem of minimizing the penalties incurred by the rejected tasks is equivalent to the Order Acceptance Problem. Shabtay's survey paper (Shabtay et al. 2013) offers a unified framework for off-line scheduling with rejection, demonstrating a close connection to scheduling with controllable processing times and scheduling with due date assignment. To our knowledge, none of these techniques considers mixed-criticality as it is defined in our paper. Mixed-criticality allows one to skip the Lo-task when it is needed for the completion of the Hi-task, but the problem formulation still incorporates the decision and optimization part.

1.3 Organization and contributions

In this section, we have shown how message retransmission in a time-triggered network can be seen as a mixed-criticality match-up scheduling problem. In Sect. 2, we give a formal description of this new problem while considering a uniprocessor environment, non-preemptive tasks, release dates, deadlines, and a general number of criticality levels. An integer linear programming (ILP) formulation is presented in Sect. 3. In Sect. 4, we use the polynomial reduction from the 3-partition problem to show strong $\mathcal{NP}$-hardness of the problem with two criticality levels even if the release dates and deadlines are omitted.

2 Problem statement

2.1 The scheduling problem

While extending standard $\alpha|\beta|\gamma$ notation and following some of the terminology introduced in Baruah et al. (2010), we can denote the scheduling problem we study as $\mathbf{1}|\mathbf{r_i}, \tilde{\mathbf{d}}_\mathbf{i}, \mathbf{mc} = \mathcal{L}, \mathbf{mu}|\mathbf{C}_{\max}$, where $mc = \mathcal{L}$ stands for mixed-criticality scheduling with $\mathcal{L}$ levels and *mu* stands for the match-up. An instance of the mixed-criticality problem is specified as a finite collection of tasks: $I_{MC} = \{T_1, \ldots, T_n\}$. Task T_i is given by a 4-tuple of parameters: $(r_i, \widetilde{d}_i, \chi_i, P_i)$, where

- $r_i \in \mathbb{Q}_+$ denotes the release date (called an *offset* in the real-time community),
- $\widetilde{d}_i \in \mathbb{Q}_+$ denotes the deadline,
- $\chi_i \in \{1, 2, \ldots, \mathcal{L}\}$ is the task criticality. The larger the value, the higher the criticality, and
- $P_i \in \mathbb{Q}_+^{\mathcal{L}}$ is vector $(P_i^1, \ldots, P_i^{\mathcal{L}})$, where ℓth entry P_i^ℓ is the processing time of task T_i at criticality level $\ell \leq \chi_i$. The P_i^ℓ is undefined for $\ell > \chi_i$. We assume that P_i is a non-decreasing function of the criticality level, i.e., $P_i^\ell \leq P_i^\jmath \ \forall \ell, \jmath \in \{1, 2, \ldots, \chi_i\} : \ell \leq \jmath$.

Furthermore, we formulate the scheduling problem as follows:

- Let s_i be the start time of task T_i.
- Let ℓ-schedule be the schedule of the criticality level $\ell \in \{1, 2, \ldots, \mathcal{L}\}$. Note that there is just one s_i of T_i in all ℓ-schedules with $\ell \in \{1, 2, \ldots, \chi_i\}$.
- The ℓ-schedule is feasible iff every task T_i with $\chi_i \geq \ell$ receives P_i^ℓ units of non-preemptive resource time (i.e., for all $j \neq i$, every task T_j with $\chi_j \geq \ell$ verifies the inequality $s_i + P_i^\ell \leq s_j$ or the inequality $s_j + P_j^\ell \leq s_i$) from r_i to $\widetilde{d}_i$ (i.e., $s_i \geq r_i$ and $s_i + P_i^\ell \leq \widetilde{d}_i$). The schedule is feasible iff all ℓ-schedules for $\ell \in 1, 2, \ldots, \mathcal{L}$ are feasible.
- Let $C_i^\ell = s_i + P_i^\ell$ be the completion time of task T_i in the ℓ-schedule. Please notice that there are different completion times of T_i in different ℓ-schedules.
- Let $C_{\max}^\ell$ be the maximum completion time of the ℓ-schedule, i.e., $C_{\max}^\ell = \max_{i:\chi_i \geq \ell} s_i + P_i^\ell$.

The objective is to find the start times $s_1, \ldots, s_n$, such that the schedule is feasible and $C_{\max} = \max_{\ell \in 1,2,\ldots,\mathcal{L}} C_{\max}^\ell$ is minimized.

2.2 On-line execution with match-up

The on-line execution with match-up of the schedule is the following:

Let function $e(t) \in \{1, 2, \ldots, \mathcal{L}\}$ be the execution level at time t, this function is defined on the interval $[0, C_{\max}]$. It becomes revealed only by actually executing the tasks. Function $e(t)$ defines the criticality level on which the schedule was executed at time t as follows:

- At the origin $e(0) = 1$.
- For $i \in \{1, \ldots, n\}$, while executing task T_i on execution level e at time instant $(s_i + P_i^\ell - \varepsilon)$ the execution level at time instant $(s_i + P_i^\ell)$ is
  - set to 1 if task T_i is completed (i.e., it matches up with the original schedule),
  - incremented if task T_i is not completed and $e(s_i + P_i^\ell - \varepsilon) < \chi_i$,
  - undefined if task T_i is not completed and $e(s_i + P_i^\ell - \varepsilon) = \chi_i$ (such behavior is supposed to be erroneous by definition).
- At other time instants, the execution level does not change its value.

If $e(s_i) = 1$ then task T_i is started at s_i, otherwise task T_i is skipped.

The execution level function may have a different shape on the time interval $[0, C_{\max}]$ during different on-line executions of the schedule.

Example Let us consider a time-triggered network transmitting seven messages, which have a specific release date, deadline, and criticality. The messages are transmitted periodically with a period of 100 time units. The messages are to be transmitted within one period, which also defines the maximum deadline.

For example, message 1 has a release date 0, deadline 100, and criticality 4. The increase of criticality level is used to model a message retransmission as follows. The transmission of message 1 takes 3 time units, and the transmission of the

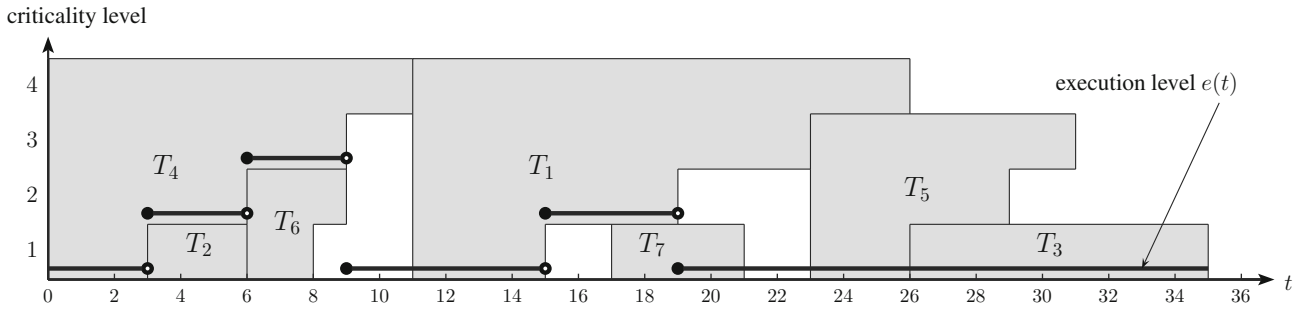

Fig. 2 An optimal schedule for the I_{MC} instance and execution level $e(t)$ of one possible on-line behavior

corresponding acknowledgment takes 1 time unit. The task is completed when the acknowledgment is positive, otherwise the message is retransmitted and a new acknowledgment is sent (both within 4 time units). The last retransmission does not need to be acknowledged, since there is no action to be taken anyway.

Consequently, we formulate the following instance I_{MC} with four criticality levels and seven tasks. Task $T_i = (r_i, \widetilde{d}_i, \chi_i, [P_i^1, P_i^2, P_i^3, P_i^4])$ is given for $i = 1 \ldots 7$ as follows:

$T_1 = (0, 100, 4, [4, 8, 12, 15])$, $T_2 = (0, 10, 1, [3, -, -, -])$, $T_3 = (0, 100, 1, [9, -, -, -])$, $T_4 = (0, 100, 4, [3, 6, 9, 11])$, $T_5 = (20, 32, 3, [3, 6, 8, -])$, $T_6 = (0, 100, 2, [2, 3, -, -])$, $T_7 = (8, 21, 1, [4, -, -, -])$.

The schedule is shown in Fig. 2, where the MC schedule consists of the tasks in the form of a letter F instead of traditional rectangles, representing tasks with one processing time only. The bold line is used to show the execution level $e(t)$ of one possible on-line behavior. In this case, task T_4 has a duration of 9, and consequently, T_2 and T_6 are skipped. Task T_1 has a duration of 8, and T_7 is skipped as well. Finally, tasks T_5 and T_3 are executed on level 1.

3 Integer linear programming formulation

In this section, we formulate the previously defined $\mathbf{1|r_i, \tilde{d}_i, mc} = \mathcal{L}, \mathbf{mu|C}_{\max}$ problem by integer linear programming. Let x_{ij} be a binary decision variable such that $x_{ij} = 1$ if and only if task T_i precedes task T_j in the schedule. The rest of the variables map to the corresponding parameters of a task that have been defined in the previous section.

The (mixed) integer linear program for our problem is as follows:

$$\min C_{\max} \tag{1}$$

subject to:

$$s_i \geq r_i \qquad \forall i \in \{1, \ldots, n\} \tag{2}$$

$$s_i + P_i^{\chi_i} \leq \widetilde{d}_i \qquad \forall i \in \{1, \ldots, n\} \tag{3}$$

$$s_i + P_i^{\chi_i} \leq C_{\max} \qquad \forall i \in \{1, \ldots, n\} \tag{4}$$

$$P_j^{\min\{\chi_i, \chi_j\}} \leq s_i - s_j + x_{ij} M \leq M - P_i^{\min\{\chi_i, \chi_j\}} \qquad \forall i, j \in \{1, \ldots, n\}, i < j \tag{5}$$

variables:

$$s_i \in \mathbb{Q}_0^+ \qquad \forall i \in \{1 \ldots n\}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in \{1, \ldots, n\}, i < j$$

$$C_{\max} \in \langle 0, M \rangle \subset \mathbb{Q}_0^+$$

The inequalities (2) and (3) say that the specific task T_i has to be executed (and completed) within the time window defined by its release date and deadline. The inequality (4) is the optimization constraint that makes the schedule as compact as possible. The double inequality (5) expresses the resource constraints. The so-called big M constant is used to ensure that only one of the two inequalities is operational for a given pair of i and j, since both corresponding constraints cannot be valid at the same time. The value of M may be any sufficiently large constant, but for our purposes, the value of $\max\{\widetilde{d}_i\}$ is sufficient. The resource constraints need to be verified only for the highest level on which T_i and T_j may collide (i.e., $\min\{\chi_i, \chi_j\}$), since P_i^ℓ is a non-decreasing function of ℓ.

4 The computational complexity of the MC scheduling problem

Our optimization problem, denoted as $1|r_i, \tilde{d}_i, mc = \mathcal{L}, mu|C_{\max}$, is $\mathcal{NP}$-hard in the strong sense, since for a single criticality level the problem reduces to the so-called *Bratley's problem* $1|r_i, \tilde{d}_i|C_{\max}$ (Błazewicz et al. 2001), which is known to be an $\mathcal{NP}$-hard problem in the strong sense due to the polynomial reduction from the 3-partition problem (Lenstra et al. 1977).

In this section we prove that our optimization problem is $\mathcal{NP}$-hard in the strong sense even if there are only two criticality levels and the time constraints are dropped, i.e., in the case of the $1|mc = 2, mu|C_{\max}$ problem.

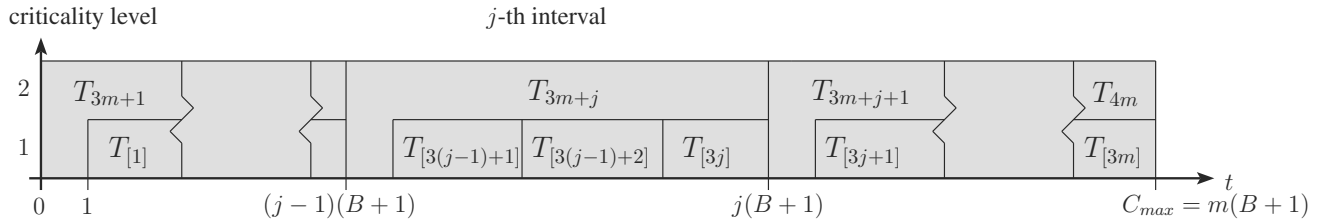

Fig. 3 Construction of the optimal schedule with $C_{\max} = m(B+1)$ from the 3-partition solution

Theorem 1 *The* $1|mc = 2, mu|C_{\max}$ *problem is* $\mathcal{NP}$*-hard in the strong sense.*

Proof **(I)** The proof is based on the polynomial reduction from the 3-partition problem.

An instance of the **3-partition problem**, I_{3P}, is given as a multiset A of $3m$ integers $a_1, a_2, \ldots, a_{3m}$ and a positive integer B such that $\forall i \in \{1, 2, \ldots, 3m\} : \frac{B}{4} < a_i < \frac{B}{2}$ and $\sum_{i=1}^{3m} a_i = mB$. The problem is to determine whether A can be partitioned into m disjoint subsets $A_1, A_2, \ldots, A_m$ such that $\forall j \in \{1, 2, \ldots, m\} : \sum_{a_i \in A_j} a_i = B$.

Note that if we show that there is a subset A_j that contains integers summing to B, then it must contain three integers. This follows from the assumption $\frac{B}{4} < a_i < \frac{B}{2}$.

From the given instance of the 3-partition problem $I_{3P} = (A, B)$, we build a mixed-criticality instance I_{MC} comprising $4m$ tasks $T_i = (\chi_i, [P_i^1, P_i^2])$ as follows:

- $\forall i \in \{1, 2, \ldots, 3m\} : T_i = (1, [a_i, -])$. Each of these Lo-tasks T_i *corresponds to* the element a_i of I_{3P}.
- $\forall i \in \{3m+1, 3m+2, \ldots, 4m\} : T_i = (2, [1, B+1])$. These are "additional/artificial" Hi-tasks used to separate the subsets.

(II) If $I_{3P} = (A, B)$ has a solution, then the optimal solution of the related I_{MC} has value of $C_{\max} = m(B+1)$.

We prove this statement as follows. Let us construct the schedule as follows: first, we partition the time axis while scheduling each Hi-task T_{3m+j} at the start of the jth interval $[(j-1)(B+1), j(B+1))$, hence $s_{3m+j} = (j-1)(B+1)$ for all $j \in \{1, 2, \ldots, m\}$. This task needs the whole interval $[(j-1)(B+1), j(B+1))$ to complete its execution in the Hi-schedule at $C^2_{3m+j} = j(B+1)$ because its $P^2_{3m+j} = B+1$. Therefore, $C^2_{\max} = m(B+1)$, and the Lo-schedule is partitioned into m intervals.

Then we can schedule the Lo-tasks corresponding to $a_i \in A_j$ into the jth interval of the Lo-schedule. This is possible because each interval has a gap of B time units left for the Lo-tasks and instance $I_{3P} = (A, B)$ has a solution. Finally, $C^1_{\max} = m(B+1)$.

Figure 3 illustrates the proof of the above-mentioned lemma while reindexing the Lo-tasks up to their order in the schedule, i.e., the first Lo-task in the schedule is labeled $T_{[1]}$.

(III) If the optimal solution of the related I_{MC} has the value of $C_{\max} = m(B+1)$, then $I_{3P} = (A, B)$ has a solution.

We prove this statement as follows. If $C^2_{\max} = m(B+1)$, then the Hi-tasks need to be scheduled side by side, i.e., $s_{3m+k} = (j-1)(B+1)$, where k is the index of the jth Hi-task in the Hi-schedule and $j \in \{1, 2, \ldots, m\}$. In the Lo-schedule, there are m blocks of Lo-tasks. Each block has a processing time $\leq B$ because $C^1_{\max} = m(B+1)$ and the Hi-tasks have a total processing time m in the Lo-schedule. As the sum of the total processing time of these $3m$ Lo-tasks is equal to mB, each block must have the processing time B. Thus, if there is a way to split the Lo-tasks into these blocks, then a partition $a_i \in A_j$ exists such that $\sum_{a_i \in A_j} a_i = B$. Therefore, there is a solution to instance $I_{3P} = (A, B)$.

(IV) Finally, the $I_{3P} = (A, B)$ has a solution iff the optimal solution of the related I_{MC} has value of $C_{\max} = m(B+1)$. Therefore, the 3-partition problem is polynomially reducible to the $1|mc = 2, mu|C_{\max}$ problem. □

5 Summary

In this paper, we have shown the motivation for the non-preemptive mixed-criticality match-up scheduling problem arising from the area of communication protocols. We have formalized this problem referred to as $1|r_i, \widetilde{d}_i, mc = \mathcal{L}, mu|C_{\max}$, and we have proposed its ILP formulation. In our opinion, this scheduling problem opens new, interesting challenges, since the diagram representing the MC schedule consists of the tasks in the form of a letter F instead of traditional rectangles, representing tasks with one processing time only. We have shown that it is an $\mathcal{NP}$-hard problem in the strong sense even when the time constraints are dropped and only two criticality levels are considered.

References

Aktürk, M. S., Atamtürk, A., & Gürel, S. (2010). Parallel machine match-up scheduling with manufacturing cost considerations. *Journal of Scheduling*, *13*(1), 95–110.

Barhorst, J., Belote, T., Binns, P., Hoffman, J., Paunicka, J., Sarathy, P., Stanfill, J., Stuart, D., & Urzi, R. (2009). A research agenda for mixed-criticality systems. In *15th IEEE real-time and embedded technology and applications symposium*.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., et al. (2012). Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, *61*(8), 1140–1152.

Baruah, S., & Fohler, G. (2011). Certification-cognizant time-triggered scheduling of mixed-criticality systems. In: *Real-time systems symposium 2011* (pp. 3–12). IEEE Computer Society.

Baruah, S., Li, H., & Stougie, L. (2010). Towards the design of certifiable mixed-criticality systems. *Real-time and embedded technology and applications symposium, RTAS* (pp. 13–22). Washington, DC: IEEE Computer Society.

Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). *Robust optimization*. Princeton: Princeton University Press.

Bertsimas, D., Brown, D., & Caramanis, C. (2011). Theory and applications of robust optimization. *SIAM Review*, *53*(3), 464–501.

Błazewicz, J., Ecker, K., Schmidt, G., & Węglarz, J. (2001). *Scheduling computer and manufacturing processes* (2nd ed.). Berlin: Springer.

Briand, C., La, H., & Erschler, J. (2007). A robust approach for the single machine scheduling problem. *Journal of Scheduling*, *10*(3), 209–221.

Burns, A., & Davis, R. (2013). Mixed criticality systems—a review. http://www-users.cs.york.ac.uk/~burns/review.pdf. Department of Computer Science, University of York.

Fung, S. (2014). Online scheduling with preemption or non-completion penalties. *Journal of Scheduling*, *17*(2), 173–183.

Hanzalek, Z., & Pacha, T. (1998). Use of the fieldbus systems in academic setting. In: *Proceedings on real-time systems education III* (pp. 93–97).

Lemke, M., Konstantellos, A., Riemenschneider, R., Steinhoegl, W., Tsarchopaulos, P., & Cotta, J. (2012). *Workshop on mixed criticality systems*, Brussels.

Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343–362.

Nelson, A., Goossens, K., & Akesson, B. Dataflow formalisation of real-time streaming applications on a composable and predictable multi-processor SOC. *Journal of Systems Architecture* (in press).

Shabtay, D., Gaspar, N., & Kaspi, M. (2013). A survey on offline scheduling with rejection. *Journal of Scheduling*, *16*(1), 3–28.

Shabtay, D., & Steiner, G. (2007). A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, *155*(13), 1643–1666.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, *212*(1), 1–11.

Sojka, M., Pisa, P., Faggioli, D., Cucinotta, T., Checconi, F., Hanzalek, Z., et al. (2011). Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of Systems Architecture*, *57*(4), 366–382.

Soyster, A. L. (1973). Technical note on convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, *21*(5), 1154–1157.

Theis, J., & Fohler, G. (2013). Mixed criticality scheduling in time-triggered legacy systems. In: *1st workshop on mixed criticality systems, IEEE real-time systems symposium*.

Theis, J., Fohler, G., & Baruah, S. (2013). Schedule table generation for time-triggered mixed criticality systems. In: *1st workshop on mixed criticality systems, IEEE real-time systems symposium*.

Thevenin, S., Zufferey, N., & Widmer, M. (2014). Metaheuristics for a scheduling problem with rejection and tardiness penalties. *Journal of Scheduling*, *18*, 89–105.

Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: *Real-time systems symposium, RTSS* (pp. 239–243).