Mixed criticality schedulability analysis is highly intractable

Sanjoy Baruah

Introduction

The mixed criticality real-time workload model, described in Section 1 below, arises in certain safety-critical application domains that may be subject to mandatory certification requirements by statutory organizations. We show here (Section 2) that it is NP-hard in the strong sense to determine whether it is possible to successfully schedule a given system specified in this model upon a fully preemptive uniprocessor platform, such that all certification constraints are satisfied.

1 Model and definitions

In this section we formally define the mixed-criticality job model, and explain terms and concepts used throughout the remainder of this document. These definitions are illustrated by means of examples in Section 1.1; while reading the following definitions, it may occasionally be useful to refer forward to Section 1.1.

A mixed-criticality (MC) *job* is characterzed by a 4-tuple of parameters: $J_i = (A_i, D_i, \chi_i, C_i)$, where

- $A_i \in R^+$ is the release time.
- $D_i \in R^+$ is the deadline. We expect that $D_i \geq A_i$.
- $\chi_i \in N^+$ denotes the criticality of the job, with a larger value denoting greater criticality.
- $C_i : N^+ \rightarrow R^+$ specifies the worst case execution time (WCET) estimate of J_i for each criticality level. (It is reasonable to assume that $C_i(\ell)$ is monotonically non-decreasing with increasing ℓ.)

An MC *instance* is specified as a finite collection of such MC jobs: $I = \{J_1, J_2, \ldots, J_n\}$.

The MC job model has the following semantics. Each job J_i is released at time-instant A_i, needs to execute for some amount of time γ_i, and has a deadline at time-instant D_i. The values of A_i and D_i are known from the specification of the job. However, the value of γ_i is not known from the specifications of J_i, but only becomes revealed by actually executing the job until it *signals* that it has completed execution. γ_i may take on very different values during different execution runs: we will refer to each collection of values $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ as a possible *behavior* of instance I.

The *criticality level* of the behavior $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ of I is the smallest integer ℓ such that $\gamma_i \leq C_i(\ell)$ for all $i, 1 \leq i \leq n$. (If there is no such ℓ, then we define that behavior to be *erroneous*.)

A *scheduling strategy* for an instance I specifies, in a completely deterministic manner for all possible behaviors of I, which job (if any) to execute at each instant in time. A *clairvoyant* scheduling strategy knows the behavior of I — i.e., the value of γ_i for each $J_i \in I$ — prior to generating a schedule for I. By contrast, an *on line* scheduling strategy does not have a priori knowledge of the behavior of I: for each $J_i \in I$, the value of γ_i only become known by executing J_i until it signals that it has completed execution.

A scheduling strategy is *correct* if it satisfies the following criterion for each $\ell \geq 1$: when scheduling any behavior of criticality level ℓ, it ensures that every job J_i with $\chi_i \geq \ell$ receives sufficient execution during the interval $[A_i, D_i)$ to signal that it has completed execution.

Let us define an instance I to be MC schedulable if there exists a correct on-line scheduling strategy for it. The *MC schedulability problem* then determines whether a given MC instance is MC schedulable or not[1].

1.1 An example

Consider an MC instance I comprised of 4 jobs. Job J_2 has criticality level 1 (which is the lower criticality level), and the other 3 jobs have the higher criticality level 2. We specify the WCET function of each task for the two criticality levels by explicit enumeration: $[C_i(1), C_i(2)]$.

- $J_1 = (0, 3, 2, [1, 2])$
- $J_2 = (0, 3, 1, [2, 2])$
- $J_3 = (0, 5, 2, [1, 1])$
- $J_4 = (3, 5, 2, [1, 2])$

For this example instance, any behavior in which $\gamma_1, \gamma_2, \gamma_3$, and γ_4 are no larger than $1, 2, 1$, and 2 respectively has criticality 1; while any behavior not of criticality one in which $\gamma_1, \gamma_2, \gamma_3$, and γ_4 are no larger than $2, 2, 1$, and 2 respectively has criticality 2. All remaining behaviors are, by definition, erroneous.

S0 below denotes a possible on-line scheduling strategy for this instance I:

S0: Execute J_1 over [0,1). If J_1 has remaining execution (i.e., γ_1 is revealed to be greater than 1), then execute scheduling strategy S1 below; else,execute scheduling strategy S2 below.

S1: Execute J_1 over [1,2), J_3 over [2,3), and J_4 over [3,5).

S2: Execute J_2 over [1,3), J_3 over [3,4), and J_4 over [4,5).

Scheduling strategy S0 is not correct for I, as can be seen by considering the schedule that generates on the behavior $(1, 2, 1, 2)$. This particular behavior has criticality 2 (since γ_4, at 2, is greater than $C_4(1)$ which has value 1, it is not criticality 1); hence, a correct schedule would need to complete jobs J_1, J_3 and J_4 by their deadlines. However, the schedule generated by this scheduling strategy would have executed J_4 for only one unit by its deadline. In fact, it turns out that instance I is not MC schedulable.

2 The Intractability of MC schedulability

In this section, we categorize the computational complexity of the MC schedulability problem. We prove (Theorem 1 below) that the MC schedulability problem — given an MC instance, determine whether it is MC-schedulable — is highly intractable: NP-hard in the strong sense. (Indeed, we will see that this hardness result holds even in the highly restricted case where all the jobs in the MC instance have the same arrival times, and each job's criticality level is either 1 or 2.) This intractability implies that under the assumption that P $\neq$ NP, there can be no polynomial or pseudo-polynomial time algorithm for solving the MC schedulability problem (even in the restricted case of equal arrival times and only two criticality levels).

[1] Another problem — the *scheduling strategy verification problem* — verifies whether a given scheduling strategy is correct for a given problem instance. We will not discuss the scheduling strategy verification problem much in this document, since a thorough analysis requires us to first agree on what constitutes an acceptable representation of a scheduling strategy.

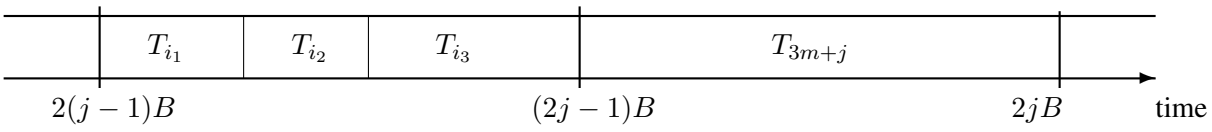

Figure 1. Scheduling strategy over the time interval $(2(j-1)B, 2jB)$. The jobs J_{i_1}, J_{1_2} and J_{i_3} satisfy $C_{i_1}(1) + C_{i_2}(1) + C_{i_3}(1) = B$; i.e., $\{s_{i_1}, s_{i_2}, s_{i_3}\}$ could be one of the partitions S_j of S. Job J_{3m+j} is executed if and only if all 3 of J_{i_1}, J_{1_2} and J_{i_3} complete execution by time-instant $(2j-1)B$; else, it is discarded and J_{i_1}, J_{1_2} and J_{i_3} executed to completion instead.

Theorem 1 *Determining MC schedulability is NP hard in the strong sense.*

Proof Sketch: This hardness is demonstrated by reducing the 3-partition problem [1], which is known to be NP-complete in the strong sense, to MC-schedulability. The 3-partition problem is defined as follows. *Given* a multiset S of $3m$ positive integers $s_0, s_1, \ldots, s_{3m-1}$ and a positive integer B such that $B/4 < s_i < B/2$ for each i and $\sum_{i=0}^{3m-1} s_i = mB$, *determine whether* S can be partitioned into m disjoint sets $S_0, S_1, \ldots, S_{m-1}$ such that, for $0 \leq j < m$, $\sum_{s_i \in S_j} s_i = B$ (note that each S_j must therefore contain exactly three elements from S).

From a given instance $I_{3P} = (S, B)$ of 3-Partition, we will build an MC instance I_{MC} comprised of $4m$ jobs, as follows:

- For each i, $0 \leq i < 3m$, job J_i has the following parameters: $A_i = 0$; $D_i = 2mB$; $\chi_i = 2$; $C_i(1) = s_i$ and $C_i(2) = 2s_i$. (We will say that the job J_i of I_{MC} *corresponds to* the element s_i of I_{3P}.)
- For each j, $0 \leq j < m$, job J_{3m+j} has the following parameters: $A_{3m+j} = 0$; $D_{3m+j} = 2(j+1)B$; $\chi_{3m+j} = 1$; and $C_{3m+j}(1) = C_{3m+j}(2) = B$.

If we know beforehand whether a particular behavior of I_{MC} is of criticality level 1 or 2, we could easily construct a schedule:

- The schedule for a criticality-level 2 behavior would simply discard all the jobs J_{3m+j}, $0 \leq j < m$; the remaining jobs, all of criticality level 2, have a common release time at 0 and a common deadline at $2mB$, and their total execution requirement is equal to $\sum_{i=o}^{3m-1} 2s_i = 2mB$, which equals the length of the interval over which they must be scheduled.
- The schedule for a criticality-level 1 behavior would, for each $j, 0 \leq j < m$, schedule job J_{3m+j} over the B time-units immediately preceding its deadline (i.e., over the interval $[(2j-1)B, 2jB)$, thereby ensuring that they all meet their deadlines. The total amount of execution thus needed is equal to mB. The remaining jobs have a common release time at 0 and a common deadline at $2mB$, and their total execution requirement is equal to $\sum_{i=o}^{3m-1} s_i = mB$. Since the jobs J_{3m+j}, $0 \leq j < m$, use mB units of execution over the interval of length $2mB$ between these jobs' common deadline and release time, there is therefore sufficient capacity available for them all to complete by their common deadline.

Hence a clairvoyant scheduler – one that knew beforehand whether a particular run would yield a behavior of criticality level one or two – would easily be able to schedule I_{MC}. An on-line scheduler, however, does not *a priori* know whether a given run-time behavior will be of criticality level one or of criticality level two; rather, this information is revealed on line during run-time by having some job J_i that receives $C_i(1)$ units of execution not signal that it has completed execution.

We will now show that ***instance $I_{3P} = (S, B)$ is in 3-Partition if and only if instance I_{MC} is MC schedulable***. (We only provide a sketch of a proof here; a formal proof appears in the appendix.)

Informally speaking, the "template" schedule shown in Figure 1 illustrates the equivalence between the 3-partition instance I_{3P} and the MC instance I_{MC}. For each j, $0 \leq j < m$, over the time-interval $[2(j-1)B, 2jB)$ we aim to complete the execution of exactly these jobs: (i) the job J_{3m+j} (of criticality-level χ_{3m+j} equal to 1), and (ii) all the jobs J_i, of criticality level $\chi_i = 2$, such that $s_i \in S_j$ (i.e., all the jobs that correspond to the elements in the j'th partition of S). To understand how this is accomplished, let us consider a particular run-time behavior of I_{MC}. Suppose that the criticality level of this behavior has not been determined to exceed 1 up until the time-instant $(2(j-1)B$; i.e., any job J_i that has received $C_i(1)$ units of execution will have signalled that it has completed execution.

Over the time interval $[2(j-1)B, 2jB)$,

1. We start out "reserving" the last B units of time over this interval — the interval $[(2j-1)B, 2jB)$ — for job J_{3m+j}. Since this job has its deadline at time-instant $2jB$ and an execution requirement equal to B, this is adequate to ensure that this job will complete by its deadline if needed.

2. This leaves B units of execution over $[2(j-1)B, (2j-1)B)$. During this interval, we will attempt to execute as many jobs J_i with $\chi_i = 2$ (i.e., of criticality-level 2) as possible, each for exactly $C_i(1)$ time units. The rationale for doing this is as follows. If any such job does not signal that it has completed execution after receiving $C_i(1)$ units of execution, we *know* that this behavior is of criticality level 2 and that jobs of criticality-level 1 are therefore not required to complete by their deadlines. Hence, we can release the capacity we had reserved in step 1 above for the criticality-level 1 job J_{3m+j}. If on the other hand all such jobs do signal that they have completed execution, we know that, even in the event of our later discovering that the behavior is of criticality-level 2, we will not need to execute these particular jobs any further (since they will have already signalled that they have completed execution). We hence know that the "savings" in execution requirement, versus the execution they would have required in the event that all jobs execute to their WCET's estimated at a level of assurance consistent with criticality level 2, is equal to the sum, over all such jobs J_i, of $(C_i(2) - C_i(1))$.

3. Now suppose that we are able to completely fill the interval $[2(j-1)B, (2j-1)B)$ (equivalently, we have identified a partition S_j of I_{3P} with the items summing to exactly B). If all these jobs signal completion, their cumulative savings versus their criticality-level 2 execution requirements is equal to B, and this is the capacity reserved in Step 1 above, and now used, for executing job J_{3m+j}. If however one or more of them do not signal completion, then their cumulative additional execution requirement may be as large as B. But we can give them this additional execution requirement by reclaiming the B units reserved for the criticality-level 1 job J_{3m+j} in step 1 above (since we are in this case not obligated to meet the deadlines of jobs of criticality-level 1).

4. If we are *not* able to completely fill the interval $[2(j-1)B, (2j-1)B)$ by executing criticality-level 2 jobs J_i for exactly $C_i(1)$ units each (equivalently, we are unable to identify a partition S_j with items summing to exactly B), on the other hand, then the savings in level-2 execution requirement that we will have identified is strictly less than B units. The schedule now has two choices, complete the execution of J_{3m+j} or not. Depending on which choice it makes, we identify below a potential behavior for which this schedule would not be correct, thereby allowing us to conclude that a failure to identify a 3-partition for I_{3P} implies that I_{MC} is not MC-schedulable.

    - If the schedule does not complete the execution of J_{3m+j}, then the criticality level of this behavior will remain equal to 1. The failure to meet J_{3m+j}'s deadline therefore indicates a failure to successfully schedule I_{MC}.

```
1  crit ← 1 ▷ Will be set to 2 if any J_i executes for > C_i(1) units
2  for j ← 0 to (m − 1) do
       ▷ Schedule over the time-interval [2jB, (2j + 2)B)
3      if (crit equals 1) then
4          Schedule each of the three jobs J_i such that s_i ∈ S_j for C_i(1) = s_i units during [2jB, (2j+
           1)B)
5          if (all these jobs complete execution) then
6              Schedule job J_{3m+j} over the interval [(2j + 1)B, (2j + 2)B)
           else ▷ some job J_i needs more than C_i(1) execution
7              crit ← 2
8              Schedule the three jobs J_i such that s_i ∈ S_j for s_i additional units each over the
               time-interval [(2j + 1)B, (2j + 2)B)
       else ▷ crit equals two
9          Schedule the three jobs J_i such that s_i ∈ S_j for C_i(2) = 2s_i units each over the time-
           interval [2jB, (2j + 2)B)
   end for
```

Figure 2. Constructing a scheduling strategy from a 3-partitioning

- If the schedule completes the execution of J_{3m+j}, then the criticality level of this behavior will become equal to 2 — some criticality-level 2 job J_i that has executed for strictly less that $C_i(1)$ units of execution thus far will request $C_i(2)$ units of execution. It may be verified that due to the execution of J_{3m+j}, there is not enough remaining execution capacity available to meet all jobs' deadlines.

We thus see that a failure to identify a partition S_j is equivalent to I_{MC} not being MC-schedulable.

■

Acknowledgements. Many thanks to Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Jose Verschae for very useful feedback regarding the presentation of this proof.

References

[1] GAREY, M., AND JOHNSON, D. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing 4* (1975), 397–411.

Appendix

A Proof

We now provide a detailed proof of the correctness of Theorem 1. Recall the reduction from 3-Partition to MC schedulability in the proof of Theorem 1: From a given instance $I_{3P} = (S, B)$ of 3-Partition with $B = \{s_0, s_1, \ldots, s_{3m-1}\}$, we had built an MC instance I_{MC} comprised of $4m$ jobs, as follows:

- For each i, $0 \leq i < 3m$, job J_i has the following parameters: $A_i = 0$; $D_i = 2mB$; $\chi_i = 2$; $C_i(1) = s_i$ and $C_i(2) = 2s_i$. (We will say that the job J_i of I_{MC} *corresponds to* the element s_i of I_{3P}.)
- For each j, $0 \leq j < m$, job J_{3m+j} has the following parameters: $A_{3m+j} = 0$; $D_{3m+j} = 2(j+1)B$; $\chi_{3m+j} = 1$; and $C_{3m+j}(1) = C_{3m+j}(2) = B$.

1. Initialize the variable $j \leftarrow 0$

2. Schedule exactly three criticality-2 jobs, with a cumulative criticality-1 WCET equal to B (and hence, cumulative criticality-2 WCET equal to $2B$) over the interval $[2jB, (2j+1)B)$, each for exactly its WCET estimated at criticality level 1.

3. If any of these three jobs does not complete execution, we know that the current behavior has criticality level 2; we may hence discard all remaining criticality-1 jobs, and complete the execution of the three criticality-2 jobs selected in Step 2 above. If, on the other hand, all three jobs do complete their execution, then we execute the criticality-1 job J_{3m+j} over the interval $[(2j+1)B, (2j+2)B)$ thereby enabling it to meet its deadline.

4. If j equals $(m-1)$ we are done; otherwise, $j \leftarrow (j+1)$ and we repeat the above steps, starting at step 2.

Figure 3. Any successful scheduling strategy for I_{MC} must look like this.

Lemma 1 *If $I_{3P} = (S, B)$ is in 3-Partition then I_{MC} is MC schedulable.*

Proof: Let us assume that I_{3P} is in 3-Partition, and let $S_1, S_2, \ldots, S_m$ denote a 3-partition of S. We may use this 3-partition to obtain a scheduling strategy for the jobs in I_{MC}. For each j, $0 \leq j < m$, we will construct the schedule over the interval $[2jB, (2j+1)B)$ as follows:

S1: Schedule the jobs J_i corresponding to each $s_i \in S_j$ for $C_i(1) = s_i$ units each over the time-interval $[2jB, (2j+1)B)$ (note that this is possible, since it is assumed that $\sum_{s_i \in S_j} s_i = B$).

S2: If all the jobs scheduled in step S1 above signal that they have completed execution, schedule job J_{3m+j} over the interval $[(2j+1)B, (2j+2)B)$. Else, discard job J_{3m+j} and complete the execution of the jobs begun in step S1 above. Note that this is possible since their total execution requirement at criticality level 2 is equal to $2B$, which is equal to the length of the interval $[2jB, (2j+1)B)$, all of which is now devoted to their execution.

Hence over each such interval, we will have executed all jobs corresponding to the elements in S_j for their requested amount of execution. If these jobs' execution is consistent with the behavior being of criticality level 1, then the job J_{3m+j} also gets its needed amount of execution; if not, then J_{3m+j} receives no execution (but this is alright since the behavior is revealed to be of criticality level 2 whereas $\chi_{3m+j} = 1$).

Repeating this argument for all j, $0 \leq j < m$, we will have thus successfully scheduled all the jobs in I_{MC}. The lemma follows.

(This scheduling strategy, along with some optimizations that allow us to ignore the criticality-1 jobs once we have identified that the behavior is of criticality level 2, is presented in pseudo-code form in Figure 2.) ■

Lemma 2 *If I_{MC} is MC schedulable then $I_{3P} = (S, B)$ is in 3-Partition.*

Proof: Let us suppose that I_{MC} is MC schedulable; we will use this to conclude that there must exist a 3-partition $S_1, S_2, \ldots, S_m$ for S.

Since I_{MC} is assumed to be MC schedulable, *all* its behaviors must be successfully scheduled by some optimal on-line scheduling algorithm. We claim that any successful on-line scheduling strategy must essentially behave as depicted in Figure 3. To show that this is the only successful scheduling strategy, consider a strategy that does not follow it, and let j' denote the smallest value of j at which it deviates from the above. Consider the behavior(s) of

I_{MC} in which all criticality-2 jobs that were executed over the interval $[0, (2j'+1)B)$ have executed for no more than their criticality-1 WCET's, indicating that the behavior does not (yet) have criticality-level 2.

Since the on-line scheduling strategy is assumed to differ from the one in Figure 3 in the scheduling decisions made over the time-interval $[(2j'B, (2j'+2)B)$, and Figure 3 would schedule 3 criticality-2 jobs for exactly their criticality-1 WCET's over $[2j'B, (2j'+1)B)$, it must be the case that this scheduling strategy must have scheduled 2 or fewer criticality-2 jobs for exactly their criticality-1 WCET's over this interval[2]. Consider the behavior of I_{MC} in which each such criticality-2 job that executed for exactly its criticality-1 WCET signals that it has completed execution. Since there are two or fewer such jobs and each has criticality-1 WCET strictly less than $B/2$, their cumulative criticality-1 WCET must be strictly less than B. (Equivalently, their cumulative criticality-2 WCET's is strictly less than $2B$.)

Observe that the scheduling strategy must execute job $I_{3m+j'}$ for B units, since if it did not do so the behavior in which no job in the future executes for more than its criticality-1 WCET would have criticality level of equal to one, and the schedule a failure since it failed to complete $J_{3m+j'}$ on time.

Let us now consider the behavior in which all remaining jobs – those that have not completed execution by time-instant $(2j'+2)B$ – execute for their criticality-2 WCET's. (This means that this behavior is of criticality level 2, and that the criticality-level 1 jobs do not need to meet their deadlines.) Let us determine how much execution must be completed by the common deadline of $2mB$, in order to be able to meet the deadlines of all jobs of criticality-level 2.

- At time $t = 0$, the total WCET at all criticality level 2 jobs for is equal to $\sum_{i=1}^{3m} C_i(2) = \sum_{i=1}^{3m} 2s_i = 2mB$.
- For each $j < j'$, the scheduling strategy is assumed to behave identically to the one depicted in Figure 3. Hence for each such j, 3 criticality-2 jobs, with a cumulative criticality-2 WCET equal to $2B$, will have signalled completion.
- This leaves $(2mB - 2(j'-1)B)$ units of criticality-2 execution over $[2j'B, 2mB)$. Out of this, we have seen that strictly less than $2B$ is accounted for over the interval $[2j'B, (2j'+2)B)$, by the jobs that signal completion; hence, strictly more than $(2mB - 2j'B)$ units of execution are required over the interval $[2j'B, 2mB)$.

But this remaining execution exceeds the length of the interval, and hence it is not possible that all the jobs of criticality level 2 will complete execution by their common deadline of $2mB$. This contradicts our assumption that a scheduling strategy can be optimal and different from the scheduling strategy depicted in Figure 3; equivalently, *any optimal scheduling strategy for I_{MC} must be as depicted in Figure 3*.

It is straightforward to obtain a 3-partition of S from the schedule generated by the scheduling strategy of Figure 3 on the behavior in which no job executes for more than its criticality-1 WCET: for each j, $0 \leq j < m$, S_j is comprised of the elements corresponding to the jobs scheduled by the scheduling strategy of Figure 3 over the interval $[2jB, (2j+1)B)$.

■

[2]The possibility that 4 or more criticality-2 jobs are executed for exactly their criticality-1 WCET's is ruled out by the 3-partition constraint that $B/4 < s_i < B/2$ for each i; hence, there isn't enough execution available with this B-length interval to accommodate more than 3 jobs's criticality-1 WCET's.