01 Dec 2016

Mixed-Criticality Scheduling to Minimize Makespan

Sanjoy K. Baruah

Arvind Easwaran

Zhishan Guo
Missouri University of Science and Technology, guozh@mst.edu

Recommended Citation

S. K. Baruah et al., "Mixed-Criticality Scheduling to Minimize Makespan," *Leibniz International Proceedings in Informatics, LIPIcs*, vol. 65, pp. 7.1-7.13, Dagstuhl Research Online Publication Server, Dec 2016.
The definitive version is available at https://doi.org/10.4230/LIPIcs.FSTTCS.2016.7

Mixed-Criticality Scheduling to Minimize Makespan[*]

Sanjoy Baruah[1], Arvind Easwaran[2], and Zhishan Guo[3]

1 **University of North Carolina at Chapel Hill, USA**
`baruah@cs.unc.edu`
2 **Nanyang Technological University, Singapore**
`arvinde@ntu.edu.sg`
3 **Missouri University of Science and Technology, Rolla, USA**
`zsguo@cs.unc.edu`

Abstract

In the mixed-criticality job model, each job is characterized by two execution time parameters, representing a smaller (less conservative) estimate and a larger (more conservative) estimate on its actual, unknown, execution time. Each job is further classified as being either less critical or more critical. The desired execution semantics are that all jobs should execute correctly provided all jobs complete upon being allowed to execute for up to the smaller of their execution time estimates, whereas if some jobs need to execute beyond their smaller execution time estimates (but not beyond their larger execution time estimates), then only the jobs classified as being more critical are required to execute correctly. The scheduling of collections of such mixed-criticality jobs upon identical multiprocessor platforms in order to minimize the makespan is considered here.

1998 ACM Subject Classification D.4.1 Scheduling, D.4.7 Real-time systems and embedded systems

Keywords and phrases Scheduling, Mixed criticality, Identical parallel machines, Makespan minimization, Approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2016.7

1 Introduction and motivation

The problem studied in this paper has its genesis in a collaborative project between our universities-based research group and a major US defense contractor. The defense contractor is developing fleets of unmanned aerial vehicles (UAVs) that are capable of coordinating with one another autonomously in order to accomplish goals that are broadly specified at a relatively high level. The embedded computer control systems on board such UAVs are responsible for two general classes of functions:

1. safety-critical functions relating to the safe flight of the UAV – these functions are expected to be subject to mandatory certification by the US Federal Aviation Authority (FAA); and
2. mission-critical functions that enable the UAV to actually accomplish its stated mission. The mission-critical functions are not subject to certification (although our collaborator –

[*] Work supported by NSF grants CNS 1115284, CNS 1218693, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, a grant from General Motors Corp., and a Tier-2 grant (ARC9/14) from the Ministry of Education, Singapore.

the defense contractor manufacturing the UAV – will subject them to analysis using their own correctness criteria).

Systems such as this are *mixed-criticality* systems; in mixed criticality (MC) systems, functionalities of different degrees of importance (or *criticalities*) are implemented upon a common platform. As stated above, the more critical functionalities may be required to have their correctness validated to a higher level of assurance than less critical functionalities. This difference in correctness criteria may be expressed by different *Worst-Case Execution Time* (WCET) estimates for the execution of a piece of real-time code. For validating the timing correctness of critical functionalities it is desirable to use WCET estimates that are obtained using extremely conservative tools (for example, some certification standards require that particular "certified" tools based on static code-analysis be used for determining WCET of highly safety-critical code), while less critical functionalities are often validated using (less conservative) measurement-based WCET tools. Vestal [11, page 239] articulated the practical implication of such practices in this manner: "the more confidence one needs in a task execution time bound [...] the larger and more conservative that bound tends to become." He proposed that each piece of code therefore be characterized by *multiple* WCET parameters, which are obtained by analyzing the (same) piece of code using different WCET-analysis tools and methodologies. Different sets of WCET estimates are then used to validate different correctness properties. We illustrate the essence of Vestal's idea via a simple (contrived) example.

▶ **Example 1.** Consider two jobs J_1 and J_2 executing upon a shared processor, with job J_1 being more critical than J_2. Both jobs are released at time 0, and share a common deadline at time 10. Let us suppose that the WCET of J_1, as determined by a more conservative WCET tool, is equal to 5, while the WCET of J_2, as determined using a less conservative WCET tool (since J_2 is less critical), is equal to 6. Since the sum of these WCETs exceeds the duration between the jobs' common release time and their deadline, conventional scheduling techniques cannot schedule both jobs to guarantee completion by their deadlines. However, Vestal observed in [11] that

- with regards to validating the more critical functionality (e.g., from the perspective of a certification process), it may be *irrelevant* whether the less critical job J_2 completes on time or not; and
- assigning J_1's WCET parameter the value of 5 may be deemed *too conservative* for validating less critical functionalities.

Let us suppose that the WCET of J_1 is estimated once again, this time using the less conservative WCET-determination tool; J_1's WCET is determined by this tool to be equal to 3 (rather than 5). If we were now to schedule the jobs by assigning J_1 greater priority than J_2,

- In validating the more critical functionalities, we would determine that J_1 completes by time-instant 5 and hence meets its deadline.
- The validation process for less critical functionalities concludes that J_1 completes by time-instant 3, and J_2 by time-instant 9. Hence they both complete by the deadline.

We thus see that the system is deemed as being correct in both analyses, despite our initial observation that the sum of the relevant WCETs (5 for J_1; 6 for J_2) exceeds the duration between the jobs' common release time and deadline.

The idea exposed in Example 1 – that the same system, represented using more and less conservative models, may be demonstrated to satisfy different correctness criteria for functionalities of different criticalities – has been widely explored since first proposed by Vestal [11]; there is a nice review of the current state of the art in [3].

This research. Much of the prior study on mixed-criticality scheduling has focused upon the scheduling of mixed-criticality workloads that are executed upon a single processor. A few pieces of work (e.g., [2, 9, 8, 7]) have considered multiprocessor scheduling, but they have all dealt with a very different workload model: systems of *recurrent* (periodic or sporadic) *tasks* that need to meet deadlines, rather than collections of independent jobs. In this paper, we seek to initiate the study of mixed-criticality scheduling of collections of independent jobs upon multiprocessor platforms, by considering a simple multiprocessor scheduling problem for such workloads – that of scheduling a given collection of mixed-criticality jobs (each of the kind described in Example 1 above) upon a specified number of identical processors in order to minimize the makespan of the resulting schedule. Makespan minimization is one of the basic and fundamental problems studied in multiprocessor scheduling, and we are optimistic that obtaining a better understanding of this fundamental problem will facilitate the development of a more comprehensive theory of multiprocessor mixed-criticality scheduling. Although this specific mixed-criticality problem is a highly simplified version of the motivating application problem – it was obtained by applying a large number of simplifying assumptions to the actual application system under analysis – it is hoped that exposing this problem domain to the FST&TCS community will motivate further work upon less simple, but more realistic, variants.

Our results. We derive algorithms for both non-preemptive scheduling (in which a job, once it begins execution, is allowed to execute through to completion upon the same processor on which it started to execute), and preemptive scheduling (in which an executing job may be preempted during execution, and its execution resumed later upon any processor) of collections of mixed-criticality jobs to minimize makespan upon identical multiprocessor platforms. The non-preemptive problem is NP-hard, but can be solved approximately in polynomial time to any desired degree of accuracy by a polynomial-time approximation scheme. We do not yet know whether or not the preemptive version of the problem is solvable in polynomial time; we derive here a polynomial-time $\frac{4}{3}$'rds-approximation algorithm for solving it. To our knowledge, the precise computational complexity of determining the minimum makespan under preemptive scheduling remains open.

2 System model

In this section we formally define the semantics of the mixed-criticality model and specify the problem that we are seeking to solve. An instance I of the scheduling problem we consider is specified as follows.

1. A collection $\mathcal{J}$ of n mixed-criticality jobs $J_1, J_2, \ldots, J_n$. Each job J_i is characterized by the parameters (χ_i, c_i^L, c_i^H), with $\chi_i \in \{\text{LO}, \text{HI}\}$ and $c_i^L \leq c_i^H$. The χ_i parameter denotes the *criticality* of job J_i; a job J_i with $\chi_i = \text{LO}$ is called a LO-criticality job, and one with $\chi_i = \text{HI}$ is called a HI-criticality job. The parameters c_i^L and c_i^H are the LO-*criticality WCET estimate* and the HI-*criticality WCET estimate* of job J_i; since the LO-criticality WCET estimates are assumed to be made using a less conservative tool than the HI-criticality WCET estimates, we require that $c_i^L \leq c_i^H$ for all $J_i \in \mathcal{J}$. (For LO-criticality jobs, we assume that $c_i^L = c_i^H$.)
2. A number m of unit-speed processors upon which the jobs in $\mathcal{J}$ are to to be executed.

Some additional notation: let $\mathcal{J}_H \subseteq \mathcal{J}$ denote the collection of all the jobs $J_i \in \mathcal{J}$ for which $\chi_i = \text{HI}$, and $\mathcal{J}_L \subseteq \mathcal{J}$ denote the collection of all the jobs $J_i \in \mathcal{J}$ for which $\chi_i = \text{LO}$.

System *behavior*. Our mixed-criticality model has the following semantics. Each job J_i is released at time 0, and needs to execute for a total duration γ_i. This execution must be sequential, meaning J_i is not allowed to simultaneously execute on more than one processor. The value of γ_i is not known prior to running time; it can only be discovered by actually allowing J_i to execute until it *signals* that it has completed execution. These values $\langle \gamma_1, \gamma_2, \ldots, \gamma_n \rangle$ upon a particular execution of the collection of jobs $\mathcal{J}$ together define the kind of *behavior* exhibited by $\mathcal{J}$ during that execution.

- If $\gamma_i \leq c_i^L$ for each i (i.e., each J_i signals completion without exceeding c_i^L units of execution), $\mathcal{J}$ is said to have exhibited LO-*criticality behavior*.
- If $c_i^L < \gamma_i \leq c_i^H$ for any i (i.e., some job J_i only signals completion upon executing for more than c_i^L but no more than c_i^H units of execution), $\mathcal{J}$ is said to have exhibited HI-*criticality behavior*.
- If $c_i^H < \gamma_i$ for any i (i.e., some job J_i does not signal completion despite having executed for c_i^H units), $\mathcal{J}$ is said to have exhibited *erroneous behavior*.

Correctness criteria. We define an algorithm for scheduling mixed-criticality instances to be correct if it is able to schedule any instance in such a manner that (i) during all LO-criticality behaviors of the instance, all jobs receive enough execution to be able to signal completion; and (ii) during all HI-criticality behaviors of the instance, all HI-criticality jobs receive enough execution to be able to signal completion. This is formally stated in the following definition:

▶ **Definition 2** (MC-correct). A scheduling algorithm for mixed-criticality instances is MC-correct if it ensures that:

- during any execution of an instance in which it exhibits LO-criticality behavior, all jobs signal completion; and
- during any execution of an instance in which it exhibits HI-criticality behavior, all HI-criticality jobs signal completion (although LO-criticality jobs may fail to do so).

We point out that upon some job failing to signal completion despite having executed for up to its LO-criticality WCET, (i) an MC-correct scheduling algorithm may immediately discard all LO-criticality jobs; and (ii) only those HI-criticality jobs that have not already signaled completion may need to execute for up to their HI-criticality WCETs – those that have already signaled completion (upon executing for $\leq$ their LO-criticality WCET) do not now need further execution.

Problem statement. Given an instance I comprising a collection $\mathcal{J}$ of n mixed-criticality jobs to be scheduled upon m unit-speed processors, obtain an MC-correct scheduling algorithm that minimizes the makespan of the resulting schedule.

Since the instance I may generate arbitrarily many different behaviors (the values of the actual running times $\langle \gamma_1, \gamma_2, \ldots, \gamma_n \rangle$) during different executions, we clarify what we mean by minimizing makespan: *we desire that the maximum makespan over all non-erroneous behaviors of the instance be minimized.*

3 MC-correct scheduling algorithms that minimize makespan

Observe that the maximum amount of execution that could be required in any LO-criticality behavior is equal to $\left(\sum_{J_i \in \mathcal{J}} c_i^L\right)$, while in any HI-criticality behavior in which each HI-criticality job executes to its HI-criticality WCET, the amount of execution that must occur is at least equal to $\left(\sum_{J_i \in \mathcal{J}_H} c_i^H\right)$. It is therefore evident that upon an m-processor platform,

$$\frac{\max\left\{\sum_{J_i \in \mathcal{J}} c_i^L, \sum_{J_i \in \mathcal{J}_H} c_i^H\right\}}{m}$$

is a lower bound on this desired makespan. An obvious upper bound on the makespan is given by

$$\sum_{J_i \in \mathcal{J}} c_i^H .$$

If we had a procedure for validating whether mixed-criticality instance I could be scheduled by some MC-correct scheduling algorithm with a makespan no larger than some specified constant, we could use bisection search (“binary search") between the upper and lower makespan bounds obtained above, in order to determine the minimum makespan to any desired degree of accuracy. In the remainder of this section, we will therefore attempt to design MC-correct scheduling algorithms that generate schedules with makespan no greater than some specified constant D.

3.1 Non-preemptive scheduling

The non-preemptive version of this problem is easily seen to be solved by transforming it to a two-dimensional *vector scheduling problem* [12], for which a PTAS is known [4, 5]. The transformation is fairly straightforward: given an instance I of the mixed-criticality scheduling problem comprising the n jobs $\mathcal{J}$ to be scheduled upon m processors with a makespan $\leq D$, we seek to partition $\mathcal{J}$ into the sub-sets $\mathcal{J}_1, \mathcal{J}_2, \ldots \mathcal{J}_m$ satisfying the constraints that for each j, $1 \leq j \leq m$,

$$\left(\sum_{J_i \in \mathcal{J}_j} c_i^L \leq D\right) \text{ and } \left(\sum_{J_i \in \mathcal{J}_j \wedge \chi_i = \text{HI}} c_i^H \leq D\right) .$$

If such a partitioning is found, then during run-time we would execute the HI-criticality jobs in $\mathcal{J}_j$ upon the j'th processor first for each j, $1 \leq j \leq m$. If each job J_i completes within c_i^L units of execution, then we execute the LO-criticality jobs in $\mathcal{J}_j$ next upon the j'th processor for each j, while if some J_i does not complete within c_i^L units of execution we simply discard the LO-criticality jobs and execute the HI-criticality jobs each to completion.

Observe that obtaining such a partitioning is equivalent to

1. First representing each job J_i by a 2-dimensional vector of size c_i^L along the first dimension, and size along the second dimension depending upon the value of χ_i: if χ_i = HI then the size along this dimension is set equal to c_i^H while if χ_i = LO then the size along this dimension is set equal to zero.
2. Next, partitioning the n vectors so obtained into m sub-sets, such that each partition sums to $\leq D$ along each of the two dimensions – this is exactly the 2-dimensional vector scheduling problem of [12].

The non-preemptive version of our scheduling problem, which is easily seen to be NP-hard (since the specialization of the problem to “regular” – non mixed-criticality – scheduling, in which all the jobs in $\mathcal{J}$ are of the same criticality, is already known to be NP-hard), is thus solvable in polynomial time to any desired degree of accuracy by a PTAS.

- Each J_i is initially executed at a constant rate ϕ_i^L
- If some J_i does not signal completion despite having received c_i^L units of execution, then
  - All LO-criticality jobs are immediately discarded, and
  - Each HI-criticality job henceforth executes at a constant rate ϕ_i^H until completion

Figure 1 Our preemptive run-time scheduling algorithm.

3.2 Preemptive scheduling

In contrast to the regular (i.e., not mixed-criticality) case, where preemptive scheduling of independent jobs to minimize makespan is easily seen to be solvable optimally in polynomial time using McNaughton's rule [10], this problem turns out to be surprisingly challenging for mixed-criticality instances. Indeed, we have not yet been able to determine whether the problem is solvable in polynomial time or not; what we have instead is a polynomial-time approximation algorithm with approximation factor 4/3 for solving this problem[1].

Given instance I and a desired makespan D, our strategy, which is based upon an algorithm called MC-Fluid [8, 1] for scheduling mixed-criticality sporadic tasks in order to meet all deadlines, is as follows. We will seek to determine a schedule for the jobs in $\mathcal{J}$ upon the m unit-speed processors under a *fluid scheduling* model, which allows for schedules in which individual jobs may be assigned a fraction ≤ 1 of a processor (rather than an entire processor, or none) at each instant in time, subject to the constraint that the sum of the fractions assigned to all the jobs do not exceed m at any instant. That is, we will determine *execution rates* ϕ_i^L and ϕ_i^H for each task τ_i such that the scheduling algorithm depicted in Figure 1 constitutes an MC-correct scheduling strategy for the jobs in $\mathcal{J}$ upon m processors. (Standard techniques are known for converting such a fluid schedule to schedules in which there is no processor-sharing; see, e.g., [6, page 116] for details.)

We will now describe how the values for the ϕ_i^L and ϕ_i^H parameters are determined. We start out defining some additional notation:

- For each job $J_i \in \mathcal{J}$, let *flow rates* f_i^L and f_i^H be defined as follows:

$$\begin{aligned} f_i^L &\overset{\text{def}}{=} c_i^L/D \\ f_i^H &\overset{\text{def}}{=} c_i^H/D \end{aligned}$$

  Intuitively, a fluid schedule in which J_i is executed at a constant rate f_i^L (f_i^H, respectively), over the interval $[0, D]$ will complete at or before time-instant D in any LO-criticality behavior (HI-criticality behavior, resp.) of the instance. It should be evident that it is necessary that f_i^L be ≤ 1 for each job J_i, and that f_i^H be ≤ 1 for each HI-criticality job J_i, if we are to be able to guarantee a makespan D.

- Various *cumulative flow requirements* are defined for $\mathcal{J}$ as follows – here, F_L^L denotes the cumulative LO-criticality flow rates of all LO-criticality jobs; F_H^L denotes the cumulative LO-criticality flow rates of all HI-criticality jobs; and F_H^H denotes the cumulative HI-criticality

[1] A trivial algorithm with approximation factor 2 can be obtained using McNaughton's rule as follows. First schedule the HI-criticality jobs based on their HI-criticality WCET estimates in the time interval $[0, D]$, and then schedule the LO-criticality jobs based on their LO-criticality WCET estimates in the time interval $[D, 2D]$.

1. Define a *scaling factor* ρ as follows:

$$\rho \leftarrow \max\Big\{\Big(\frac{F_L^L + F_H^L}{m}\Big), \Big(\frac{F_H^H}{m}\Big), \max_{J_i \in \mathcal{J}_H}\{f_i^H\}\Big\} \tag{1}$$

2. **If** $\rho > 1$ **then** declare failure; **else** assign values to the execution-rate variables as follows:

$$\phi_i^H \quad \leftarrow \quad (f_i^H/\rho) \text{ for all } J_i \in \mathcal{J}_H \tag{2}$$

$$\phi_i^L \quad \leftarrow \quad \begin{cases} \frac{f_i^L}{\phi_i^H - (f_i^H - f_i^L)} \times \phi_i^H, & \text{if } J_i \in \mathcal{J}_H \\ f_i^L, & \text{else (i.e., if } J_i \in \mathcal{J}_L) \end{cases} \tag{3}$$

3. **If**

$$\sum_{J_i \in \mathcal{J}} \phi_i^L \le m \tag{4}$$

then declare success **else** declare failure

Figure 2 Computing execution rates.

flow rates of all HI-criticality jobs:

$$F_L^L \stackrel{\text{def}}{=} \sum_{J_i \in \mathcal{J}_L} f_i^L$$
$$F_H^L \stackrel{\text{def}}{=} \sum_{J_i \in \mathcal{J}_H} f_i^L$$
$$F_H^H \stackrel{\text{def}}{=} \sum_{J_i \in \mathcal{J}_H} f_i^H$$

The following observation directly follows from the definitions of F_L^L, F_H^L, and F_H^H:

▶ **Observation 3.** *It is necessary that* $(F_L^L + F_H^L) \le m$, *and* $F_H^H \le m$, *if we are to be able to guarantee a makespan* D *for* $\mathcal{J}$ *upon* m *processors.*

As stated in Figure 1, our run-time scheduling algorithm requires that values be assigned to the execution-rate variables $\{\phi_i^L\}_{J_i \in \mathcal{J}} \bigcup \{\phi_i^H\}_{J_i \in \mathcal{J}_H}$ prior to run-time. In Figure 2 we describe, in pseudo-code form, the algorithm for computing the values of these execution-rate variables. Before proving the correctness of this algorithm (in Section 3.3), we first illustrate its application via an example.

An example. Consider the following collection of 4 mixed-criticality jobs, to be scheduled preemptively upon a 2-processor platform with a target makespan $D \leftarrow 10$.

	χ_i	c_i^L	c_i^H
J_1	HI	3	8
J_2	HI	4	7
J_3	HI	1	1
J_4	LO	5	5

The flow rates for the jobs are obtained by dividing the corresponding WCET parameters by 10 (the value of D); the cumulative flow requirements are then computed as follows:

$$\begin{aligned}
F_L^L &= f_4^L = \mathbf{0.5} \\
F_H^L &= f_1^L + f_2^L + f_3^L = 0.3 + 0.4 + 0.1 = \mathbf{0.8} \\
F_H^H &= f_1^H + f_2^H + f_3^H = 0.8 + 0.7 + 0.1 = \mathbf{1.6}
\end{aligned}$$

The scaling factor ρ is therefore

$$\begin{aligned}
\rho &= \max\Big\{\frac{0.5+0.8}{2}, \frac{1.6}{2}, \max\{0.8, 0.7, 0.1\}\Big\} \\
&= \max\big\{1.3/2, 1.6/2, 0.8\big\} \\
&= \mathbf{0.8}
\end{aligned}$$

The HI-criticality jobs J_1, J_2, and J_3, are assigned ϕ_i^H values as follows:

$$\begin{aligned}
\phi_1^H &= \frac{0.8}{0.8} = 1.0 \\
\phi_2^H &= \frac{0.7}{0.8} = 0.875 \\
\text{and } \phi_3^H &= \frac{0.1}{0.8} = 0.125
\end{aligned}$$

All the jobs are assigned ϕ_i^L values as follows:

$$\begin{aligned}
\phi_1^L &= \frac{1.0 \times 0.3}{1.0 - (0.8 - 0.3)} = 0.6 \\
\phi_2^L &= \frac{0.875 \times 0.4}{0.875 - (0.7 - 0.4)} = \frac{14}{23} < 0.61 \\
\phi_3^L &= \frac{0.125 \times 0.1}{0.125 - (0.1 - 0.1)} = 0.1 \\
\text{and } \phi_4^L &= 0.5
\end{aligned}$$

Since $\sum_{i=1}^{4} \phi_i^L < \big(0.6 + 0.61 + 0.1 + 0.5\big) = 1.81$, which is ≤ 2 (the number of processors), our algorithm declares success.

3.3 Preemptive scheduling – proof of correctness

We will now show that the preemptive scheduling algorithm described above is correct: if the execution rates are computed as specified in Figure 2 without declaring failure for a given instance I, then the schedule resulting from using these execution rates in the manner described in Figure 1 does indeed constitute an MC-correct scheduling algorithm that always generates schedules of makespan $\leq D$ upon all non-erroneous behaviors. Our proof proceeds in several steps.

1. We first prove, in Lemma 4 below, that the rate-assignment in Figure 2 is correct, by showing that the sum of the LO-criticality and the HI-criticality execution rates assigned to the jobs in Figure 2 do not exceed m, the number of available processors.
2. Next, we show in Lemma 5 that each execution rate is assigned a valid value in Figure 2: a non-negative real number that is no larger than one.
3. We then prove, in Lemma 6, correctness upon all LO-criticality behaviors, by showing that the ϕ_i^L values assigned in Figure 2 are no smaller than the corresponding f_i^L values.

4. Finally, we prove correctness upon all HI-criticality behaviors by examining the actions of the scheduling algorithm in the event that some job does not signal completion despite having executed for up to its LO-criticality WCET (which indicates that the instance is exhibiting a HI-criticality behavior rather than a LO-criticality one).

▶ **Lemma 4.** *The sum of the* LO*-criticality execution rates assigned to all the jobs, and the sum of the* HI*-criticality rates assigned to all the* HI*-criticality jobs (i.e., the jobs in $\mathcal{J}_H$), each does not exceed the number of processors m.*

Proof. It follows from Condition 4 of Figure 2 that our algorithm declares success only if the assigned LO-criticality execution rates sum to $\leq m$.

To show that the assigned HI-criticality rates also sum to no more than m, observe that

$$\sum_{J_i \in \mathcal{J}_H} \phi_i^H = \sum_{J_i \in \mathcal{J}_H} \frac{f_i^H}{\rho} \quad \text{(By Eqn 2)} \quad = \frac{1}{\rho} F_H^H \quad \text{(By definition of } F_H^H\text{)}$$

By Equation 1, $\rho \geq (F_H^H/m)$; hence

$$\frac{1}{\rho} F_H^H \leq (\frac{m}{F_H^H}) F_H^H = m$$

and the lemma is proved. ◀

▶ **Lemma 5.** *Each ϕ_i^L and ϕ_i^H is assigned a value ≤ 1 in the algorithm of Figure 2.*

Proof. Observe that Line 1 of Figure 2 assigns ρ a value $\geq f_i^H$ for all $J_i \in \mathcal{J}_H$. Since Line 2 of Figure 2 assigns each ϕ_i^H a value f_i^H/ρ, it follows that each such ϕ_i^H has a value ≤ 1, as required.

With regards to the ϕ_i^L's, the value assigned to ϕ_i^L for each $J_i \in \mathcal{J}_L$ is equal to f_i^L (and hence ≤ 1).

For each $J_i \in \mathcal{J}_H$, we will now show that $\phi_i^L \leq \phi_i^H$. It follows from the assignment of values to ϕ_i^L (Equation 3 in Figure 2) that this will hold provided (Removed the justification in the last derivation step.)

$$\begin{aligned} & \frac{f_i^L}{\phi_i^H - \left(f_i^H - f_i^L\right)} \leq 1 \\ \Leftrightarrow \quad & f_i^L \leq \phi_i^H - \left(f_i^H - f_i^L\right) \\ \Leftrightarrow \quad & f_i^H \leq \phi_i^H \end{aligned}$$

which follows from the requirement that ρ be ≤ 1 (else, the algorithm in Figure 2 would declare failure in Step 2).

We have thus shown that $\phi_i^L \leq \phi_i^H$ for each $J_i \in \mathcal{J}_H$ (i.e., the execution rate guaranteed to each HI-criticality job does not *decrease* upon identification of HI-criticality behavior). Since we saw above that all such ϕ_i^H values are ≤ 1, it follows that the ϕ_i^L variables are also assigned values ≤ 1. ◀

▶ **Lemma 6.** *The instance is scheduled with a makespan $\leq D$ in all* LO*-criticality behaviors.*

Proof. We will first prove that for each $J_i \in \mathcal{J}$

$$\phi_i^L \geq f_i^L \tag{5}$$

Let us separately consider jobs in $\mathcal{J}_L$ and $\mathcal{J}_H$. Observe that by the definition of ϕ_i^L (Equation 3),

1. For each $J_i \in \mathcal{J}_L$, $\phi_i^L = f_i^L$.
2. For each $J_i \in \mathcal{J}_H$,

$$\begin{aligned}\phi_i^L &= f_i^L \times \frac{\phi_i^H}{\phi_i^H - \left(f_i^H - f_i^L\right)} \\ &\geq f_i^L \times \frac{\phi_i^H}{\phi_i^H} \quad \text{(Since } (f_i^H - f_i^L) \geq 0) \\ &= f_i^L\end{aligned}$$

These two cases together establish that $\phi_i^L \geq f_i^L$ for all $J_i \in \mathcal{J}$; it hence immediately follows that $\phi_i^L \cdot D$, the amount of execution that would be received by J_i if it were allowed to execute at a rate ϕ_i^L over the entire duration $[0, D)$ in any LO-criticality behavior of $\mathcal{J}$, is $\geq c_i^L$. From this we conclude that the makespan in any LO-criticality behavior is $\leq D$. ◀

▶ **Lemma 7.** *The instance is scheduled with a makespan $\leq D$ in all* HI*-criticality behaviors.*

Proof. Consider any HI-criticality behavior of the instance, and let t_o denote the first time-instant at which some job does not signal completion despite having executed for its LO-criticality WCET. We will prove below that any HI-criticality job that is active (i.e., that has not yet completed execution) at time-instant t_o receives an amount of execution no smaller than its HI-criticality WCET by time-instant D.

Suppose that HI-criticality job J_i is active at time-instant t_o. Over the interval $[0, t_o)$, this job will have received an amount of execution equal to $\phi_i^L \times t_o$; since the job is still active, it must be the case that

$$t_o \leq c_i^L / \phi_i^L \tag{6}$$

Henceforth job J_i will execute at a rate ϕ_i^H. Hence for it to complete within a makespan D, it is sufficient that

$$\begin{aligned} & t_o\phi_i^L + (D - t_o)\phi_i^H \geq c_i^H \\ \Leftrightarrow\ & D\,\phi_i^H - t_o(\phi_i^H - \phi_i^L) \geq c_i^H \\ \Leftarrow\ & D\,\phi_i^H - \frac{c_i^L}{\phi_i^L}(\phi_i^H - \phi_i^L) \geq c_i^H \text{ (By Inequality 6)} \\ \Leftrightarrow\ & D\,\phi_i^H \geq \frac{c_i^L}{\phi_i^L}(\phi_i^H - \phi_i^L) + c_i^H \\ \Leftrightarrow\ & D\,\phi_i^H \geq \frac{c_i^L\phi_i^H}{\phi_i^L} - c_i^L + c_i^H \\ \Leftrightarrow\ & D \geq \frac{c_i^L}{\phi_i^L} + \left(\frac{c_i^H - c_i^L}{\phi_i^H}\right) \\ \Leftrightarrow\ & 1 \geq \frac{f_i^L}{\phi_i^L} + \left(\frac{f_i^H - f_i^L}{\phi_i^H}\right) \text{ (Dividing by } D\text{, and applying definitions of } f_i^L,\ f_i^H) \end{aligned} \tag{7}$$

Also by Equation 3, for each $J_i \in \mathcal{J}_H$ we have

$$\begin{aligned} & \phi_i^L = \frac{f_i^L\phi_i^H}{\phi_i^H - \left(f_i^H - f_i^L\right)} \\ \Leftrightarrow\ & \frac{\phi_i^H - \left(f_i^H - f_i^L\right)}{\phi_i^H} = \frac{f_i^L}{\phi_i^L}\end{aligned}$$

$$\Leftrightarrow 1 - \left(\frac{f_i^H - f_i^L}{\phi_i^H}\right) = \frac{f_i^L}{\phi_i^L}$$
$$\Leftrightarrow 1 = \frac{f_i^L}{\phi_i^L} + \left(\frac{f_i^H - f_i^L}{\phi_i^H}\right)$$

thereby establishing Condition 7 and completing the proof of the lemma. ◀

3.4 Preemptive scheduling – A 4/3'rds approximation Bound

We now prove that MC-correct scheduling algorithm described in Section 3.2 is a 4/3'rds approximate algorithm for preemptive scheduling to minimize makespan. Our approach towards showing this is as follows. A straightforward generalization of Observation 3 leads us to conclude that for mixed-criticality instance $\mathcal{J}$ to be schedulable with makespan $s \times D$ upon m processors, it is necessary that $(F_L^L + F_H^L)$ and F_H^H for the instance both be $\leq m \times s$, and that in addition $f_i^H \leq s$ for each $J_i \in \mathcal{J}_H$ and $f_i^L \leq s$ for each $J_i \in \mathcal{J}_L$. It therefore follows that the scaling factor ρ that is computed in Expression 1 of the algorithm of Figure 2 for such a system is $\leq s$. We will show below, in Lemma 9, that if $\rho \leq 3/4$ and the ϕ_i^H, ϕ_i^L values are computed as specified in Expressions 2–3 of Figure 2, then the ϕ_i^L's so computed are guaranteed to sum to $\leq m$ and therefore satisfy Condition 4 of Figure 2 (which in turn means that the system is scheduled with makespan $\leq D$ upon m processors). The approximation ratio follows, by observing that 4/3 is the multiplicative inverse of 3/4.

First, a technical lemma.

▶ **Lemma 8.** *Let c denote any positive constant. The function*

$$f(x) \overset{def}{=} \frac{x(c-x)}{\frac{c}{3}+x}$$

is $\leq \frac{c}{3}$ for all values of $x \in [0, c]$.

Proof. (This lemma is easily proved rigorously using standard techniques from the calculus; we skip the details here in favor of a high-level outline.) Taking the derivative of $f(x)$ with respect to x, we see that the only value of $x \in [0, c]$ where this derivative equals zero is $x \leftarrow c/3$. We therefore conclude that $f(x)$ takes on its maximum value over $[0, c]$ for one of the values of $x \in \{0, c/3, c\}$. Explicit computation of $f(x)$ at each of these values reveals that the value is maximized at $x = c/3$, where it takes on the value $c/3$. ◀

▶ **Lemma 9.** *If $\rho \leq 3/4$ and ϕ_i^H, ϕ_i^L values are computed as specified in Expressions 2–3 of Figure 2, then the ϕ_i^L values so computed satisfy Condition 4.*

Proof. Let us first rewrite Condition 4 to an equivalent form expressed in Condition 8 below.

$$\sum_{J_i \in \mathcal{J}} \phi_i^L \leq m$$
$$\Leftrightarrow \sum_{J_i \in \mathcal{J}_L} \phi_i^L + \sum_{J_i \in \mathcal{J}_H} \phi_i^L \leq m$$
$$\Leftrightarrow F_L^L + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L \phi_i^H}{\phi_i^H - (f_i^H - f_i^L)} \leq m$$

$$\begin{aligned}
&\Leftrightarrow F_L^L + \sum_{J_i \in \mathcal{J}_H} f_i^L \Big(1 + \frac{\big(f_i^H - f_i^L\big)}{\phi_i^H - \big(f_i^H - f_i^L\big)}\Big) \le m \\
&\Leftrightarrow F_L^L + \sum_{J_i \in \mathcal{J}_H} f_i^L + \sum_{\tau_i \in \mathcal{J}_H} \frac{f_i^L(f_i^H - f_i^L)}{\phi_i^H - \big(f_i^H - f_i^L\big)} \le m \\
&\Leftrightarrow F_L^L + F_H^L + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L\big(f_i^H - f_i^L\big)}{\phi_i^H - \big(f_i^H - f_i^L\big)} \le m \qquad (8)
\end{aligned}$$

We will show, in the remainder of this proof, that if $\rho \le 3/4$ then Condition 8 is satisfied; this will serve to establish the correctness of Lemma 9.

Let us assume henceforth that $\rho \le 3/4$. From the definition of ρ (Expression 1), it follows that

$$\begin{aligned}
F_L^L + F_H^L &\le \frac{3}{4}m & (9)\\
F_H^H &\le \frac{3}{4}m & (10)\\
\forall J_i \in \mathcal{J}_H \quad f_i^H &\le \frac{3}{4} & (11)
\end{aligned}$$

Additionally, since $\phi_i^H \leftarrow f_i^H/\rho$, it must hold that

$$\forall J_i \in \mathcal{J}_H \quad \phi_i^H \ \ge \frac{4}{3} f_i^H \qquad (12)$$

Let us use Inequalities 9–12 to further simplify Condition 8.

$$\begin{aligned}
&F_L^L + F_H^L + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L\big(f_i^H - f_i^L\big)}{\phi_i^H - \big(f_i^H - f_i^L\big)} \le m \\
&\Leftarrow \quad \frac{3}{4}m + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L\big(f_i^H - f_i^L\big)}{\phi_i^H - \big(f_i^H - f_i^L\big)} \le m \text{ (By Ineq. 9)} \\
&\Leftarrow \quad \frac{3}{4}m + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L\big(f_i^H -_i^L\big)}{\frac{4}{3} f_i^H - \big(f_i^H - f_i^L\big)} \le m \text{ (By Ineq. 12)} \\
&\Leftrightarrow \quad \frac{3}{4}m + \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L\big(f_i^H - f_i^L\big)}{\frac{f_i^H}{3} + f_i^L} \le m \\
&\Leftrightarrow \quad \sum_{J_i \in \mathcal{J}_H} \frac{f_i^L\big(f_i^H - f_i^L\big)}{\frac{f_i^H}{3} + f_i^L} \le \frac{m}{4} \\
&\Leftarrow \quad \sum_{J_i \in \mathcal{J}_H} \frac{f_i^H}{3} \le \frac{m}{4} \quad \text{(By Lemma 8)} \\
&\Leftrightarrow \quad \frac{1}{3} \cdot (F_H^H) \le \frac{m}{4} \\
&\Leftarrow \quad \Big(\frac{1}{3} \cdot \frac{3}{4}m \le \frac{m}{4}\Big) \quad \text{(By Inequality 10)} \\
&\Leftrightarrow \quad \Big(\frac{m}{4} \le \frac{m}{4}\Big)
\end{aligned}$$

and Lemma 9 is thereby proved. ◀

4 Summary and conclusions

Mixed-criticality scheduling is emerging as an increasingly important topic in the design, analysis, and implementation of safety-critical embedded systems. Most prior work on this topic has been restricted to uniprocessor scheduling; what little work has been done on multiprocessor scheduling has primarily focused upon recurrent (periodic and sporadic) workload models that are very different from the one we consider in this paper. We have adapted ideas from some such prior work, and have applied them to our problem of scheduling collections of independent jobs in order to minimize makespan. We have designed algorithms for both preemptive and non-preemptive scheduling of such workloads, but have not yet been able to classify the computational complexity of preemptive scheduling to minimize makespan – we leave this as an open problem.

We reiterate a point we had made earlier in this manuscript – although the particular problem we have presented here was obtained by applying a large number of simplifying assumptions to the actual application system under analysis, we hope that exposing this very interesting and important problem domain to the FST&TCS community will encourage members of this community to work upon more realistic, and more complex, variations.

References

1 Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo. MC-Fluid: simplified and optimally quantified. In *Real-Time Systems Symposium (RTSS), 2015 IEEE*, Dec 2015.

2 Sanjoy K. Baruah, Liliana Cucu-Grosjean, Robert I. Davis, and Claire Maiza. Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121). *Dagstuhl Reports*, 5(3):84–142, 2015. doi:10.4230/DagRep.5.3.84.

3 Alan Burns and Robert Davis. Mixed-criticality systems: A review. Available at http://www-users.cs.york.ac.uk/~burns/review.pdf, 2013.

4 Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, January 1999.

5 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004. doi:10.1137/S0097539799356265.

6 Jr. E. Coffman and P. J. Denning. *Operating Systems Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

7 Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *International Conference on Embedded Software (EMSOFT)*, pages 17:1–17:15, Montreal, Oct 2013.

8 Lee Jaewoo, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, A. Easwaran, Insik Shin, and Insup Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014.

9 Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, and Chenyang Lu. Mixed-criticality federated scheduling for parallel real-time tasks. In *Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016.

10 R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

11 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium*, pages 239–243, Tucson, AZ, December 2007. IEEE Computer Society Press.

12 Tjark Vredeveld. Vector scheduling problems. In Min-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2014.