Mixed-Criticality Scheduling of Sporadic Task Systems

Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne Ster, Leen Stougie

Mixed-Criticality Scheduling of Sporadic Task Systems

Sanjoy K. Baruah[1], Vincenzo Bonifaci[2], Gianlorenzo D'Angelo[3], Alberto Marchetti-Spaccamela[4], Suzanne van der Ster[5], and Leen Stougie[5,6]

[1] University of North Carolina at Chapel Hill, USA
baruah@cs.unc.edu
[2] Max-Planck Institut für Informatik, Saarbrücken, Germany
bonifaci@mpi-inf.mpg.de
[3] University of L'Aquila, Italy
gianlorenzo.dangelo@univaq.it
[4] Sapienza Università di Roma, Rome, Italy
alberto@dis.uniroma1.it
[5] Vrije Universiteit Amsterdam, the Netherlands
suzanne.vander.ster@vu.nl,l.stougie@vu.nl
[6] CWI, Amsterdam, the Netherlands
stougie@cwi.nl

Abstract. We consider the scheduling of mixed-criticality task systems, that is, systems where each task to be scheduled has multiple levels of worst-case execution time estimates. We design a scheduling algorithm, EDF-VD, whose effectiveness we analyze using the processor speedup metric: we show that any 2-level task system that is schedulable on a unit-speed processor is correctly scheduled by EDF-VD using speed ϕ; here $\phi < 1.619$ is the golden ratio. We also show how to generalize the algorithm to $K > 2$ criticality levels.We finally consider 2-level instances on m identical machines. We prove speedup bounds for scheduling an independent collection of jobs and for the partitioned scheduling of a 2-level task system.

1 Introduction

We study a scheduling problem occurring in safety-critical systems that are subject to certification requirements. Our work is motivated by the increasing trend in embedded systems towards integrating multiple functionalities on a common platform – consider, for example, the IMA (Integrated Modular Avionics) initiative for aerospace and AUTOSAR (AUTomotive Open System ARchitecture) for the automotive industry. Such platforms may support functionalities of different degrees of importance or *criticalities*. Some of the more safety-critical functionalities may be subject to mandatory certification by statutory certification authorities (CAs). The increasing prevalence of platform integration means that even in highly safety-critical systems, typically only a relatively small fraction of the overall system is actually of critical functionality and needs to be

certified. The definition of procedures that will allow for the cost-effective certification of such mixed-criticality systems has been identified as a challenging collection of problems [1]. As a recognition of the importance of these challenges we mention the Mixed Criticality Architecture Requirements (MCAR) program led by several federal agencies and by industry in the US, aimed at streamlining the certification process for safety-critical embedded systems with the long term goal of devising efficient and cost-effective certification processes.

During the certification process, the CA makes certain assumptions about the worst-case run-time behavior of the system. We focus on one particular aspect of run-time behavior: the worst case execution time (WCET) of pieces of code. CAs tend to be very conservative, and hence it is typically the case that the WCET-analysis tools, techniques, and methodologies used by the CA will yield WCET estimates that are far more pessimistic (i.e., larger) than those the system designer would use during the design process. On the other hand, while the CA is only concerned with the correctness of the part of the system that is subject to certification the system designer wishes to ensure that the entire system is correct, including the non-critical parts. We illustrate this by an example from the domain of unmanned aerial vehicles. The functionalities on board such vehicles may be classified into two levels of criticality:

- Level 1: the *mission-critical* functionalities, concerning reconnaissance and surveillance objectives, like capturing images from the ground, transmitting these images to the base station, etc.
- Level 2: the *flight-critical* functionalities: to be performed by the aircraft to ensure its safe operation.

For permission to operate such vehicles over civilian airspace, it is mandatory that its flight-critical (i.e., level 2) functionalities be certified by statutory CAs such as the Federal Aviation Authority (FAA) in the US, or the European Aviation Safety Agency (EASA) in Europe. These CAs are not concerned with the mission-critical functionalities, which must be validated separately by the clients and the vendor-manufacturer. The latter are also interested in the level 2 functionalities, but typically to standards that are less rigorous than the ones used by the civilian CAs.

In Section 2 we will formally define the scheduling problem that we study in this work but as an example of the previous scenario let us consider a simple instance of two jobs, J_1 of criticality level 1 and J_2 of criticality level 2. Job J_1 is characterized by a release time, a deadline and a WCET $c_1(1)$, while job J_2 is characterized by a release time, a deadline and a pair $(c_2(1), c_2(2))$ giving the WCET for level 1 and level 2, respectively – $c_2(1)$ is the WCET estimate used by the system designer, whereas $c_2(2)$ is the (typically much larger) WCET estimate used by the CA. The scheduling goal is to find a schedule such that the following conditions are both verified

1. *(Validation by client/ manufacturer).* If the execution time of J_1 is no larger than $c_1(1)$ and that of J_2 no larger than $c_2(1)$ then we require that both J_1 and J_2 complete by their deadlines.

2. *(Certification by CA).* If the execution time of J_2 is no larger than $c_2(2)$ then we require that J_2 must complete by its deadline. (Since the CA is not concerned with the level 1 job J_1, it follows that if J_2 executes for more than $c_2(1)$ but no more than $c_2(2)$, then it is not mandatory anymore to complete J_1 by its deadline.)

The difficulty in finding such a schedule is that scheduling decisions must be made *before* knowing whether we are in case 1 or 2 above: the scheduling algorithm becomes aware of J_2's actual execution time only by executing it (i.e. if the job does not finish after $c_2(1)$ execution time).

If there are more than two jobs then we require that if all jobs of level 2 meet their level 1 WCET estimates then all jobs must be completed by their deadline; otherwise we require that only level 2 jobs should meet their deadlines. This model is easily generalized to more than two criticality levels (see Sect. 2 for a formal definition);

Many real-time systems are better modeled as collections of recurrent processes (known as *sporadic tasks*) rather than as a finite set of jobs. We refer to [3] for an introduction to sporadic task systems; mixed-criticality sporadic task systems are defined in Section 2 below. We observe that schedulability analysis of such systems is typically far more difficult than the analysis of systems modeled as collections of independent jobs, since (i) a sporadic task system can generate infinitely many jobs during any one run; and (ii) the collection of jobs generated during different runs of the system may be different: in general, a single system may legally give rise to infinitely many different collections of jobs.

One focus of this paper is to study this more difficult problem of scheduling mixed-criticality systems modeled as collections of sporadic tasks, upon a single shared preemptive processor.

Related work. The mixed-criticality model that we follow was studied, for independent collections of jobs, in [4]. The authors considered a finite collection of jobs of two criticality levels to be scheduled on one machine. They analysed, using the processor speedup factor (cf. resource augmentation in performance analysis of approximation algorithms, as initiated in [6]), the effectiveness of two techniques that are widely used in practice in real time scheduling. They proposed an algorithm called OCBP and showed that OCBP has a processor speedup factor equal to the golden ratio ϕ. These results were later extended [2] to any number of levels showing also that OCBP is tight for two levels.

The mixed-criticality model has been extended to recurrent task systems in [7]; however the proposed algorithm has a pseudopolynomial running time per scheduling decision and, therefore, cannot be applied in practice.

To the best of our knowledge no results are known for multiple machines.

Our results. Our main result concerns the scheduling of a mixed-criticality task system on a single machine. We design a sufficient schedulability condition and a scheduling algorithm (EDF-VD) whose effectiveness we analyze using the processor speedup metric. We show that any 2-level task system that is schedulable by any algorithm on a single processor of unit speed is correctly scheduled by

EDF-VD on a speed ϕ processor; here $\phi < 1.619$ is the golden ratio. We then show how to generalize the schedulability condition and the algorithm to $K > 2$ criticality levels.

The other main contribution is the study of 2-level instances on multiple identical machines. We generalize the OCBP algorithm for an independent collection of jobs to the case of m machines, proving a speedup bound of $\phi+1-1/m$. Finally we consider the *partitioned* scheduling of a 2-level task system, in which the tasks are assigned to the machines and then scheduled on each machine independently, without migration; we extend the results for the single machine case to derive a $\phi + \epsilon$ speedup algorithm for any $\epsilon > 0$.

Organization of the paper. We give a formal description of the mixed-criticality model along with basic concepts and notation in Section 2. In Section 3 we treat the case of a single machine, for two (Section 3.1) and more than two (Section 3.2) criticality levels. Section 4 is devoted to the case of multiple machines; collection of independent jobs are treated in Section 4.1, while the partitioned scheduling of a task system is considered in Section 4.2.

2 Model

MC jobs. A job in a K-level MC system is characterized by a 4-tuple of parameters: $J_j = (r_j, d_j, \chi_j, c_j)$, where r_j is the release time, d_j is the deadline ($d_j \geq r_j$), $\chi_j \in \{1, 2, \ldots, K\}$ is the *criticality level* of the job and c_j is a vector $(c_j(1), c_j(2), \ldots, c_j(K))$ representing the *worst-case execution times* (WCET) of job J_j at each level; it is assumed that $c_j(1) \leq c_j(2) \leq \cdots \leq c_j(K)$ and $c_j(i) = c_j(\chi_j)$, for each $i > \chi_j$. Each job J_j in a collection $J_1, \ldots, J_n$ should receive execution time C_j within time window $[r_j, d_j]$. The value of C_j is not known but is discovered by executing job J_j until it signals completion. A collection of realized values $(C_1, C_2, \ldots, C_n)$ is called a scenario. The criticality level of a scenario $(C_1, \ldots, C_n)$ is defined as the smallest integer ℓ such that $C_j \leq c_j(\ell)$ for each job J_j. (We only consider scenarios where such an ℓ exists; i.e. $C_j \leq c_j(K)$, $\forall j$.) The crucial aspect of the model is that, in a scenario of level ℓ, it is necessary to guarantee only that jobs of criticality at least ℓ are completed before their deadlines. In other words, once a scenario is known to be of level ℓ, the jobs of criticality $\ell - 1$ or less can be safely dropped.

MC task systems. Let $T = (\tau_1, \ldots, \tau_n)$ be a system of n tasks, each task $\tau_j = (c_j, p_j, \chi_j)$ having a worst-case *execution time vector* $c_j = (c_j(1), c_j(2), \ldots, c_j(K))$, a *period* p_j, and a *criticality level* $\chi_j \in \{1, 2, \ldots, K\}$. Again we assume that $c_j(1) \leq \cdots \leq c_j(K)$. Task τ_j generates a potentially infinite sequence of jobs, with successive jobs being released at least p_j units apart. Each such job has a deadline that is p_j time units after its release (*implicit deadlines*). The criticality of such job is χ_j, and its WCET vector is given by c_j.

MC-schedulability. An (online) algorithm schedules a sporadic task system T correctly if it is able to schedule *every* job sequence generated by T such that

if the criticality level of the corresponding scenario is ℓ, then all jobs of level at least ℓ are completed between their release time and deadline. A system is called *MC-schedulable* if it admits some correct scheduling algorithm.

Utilization in task systems. Let $L_k = \{j \in \{1, \ldots, n\} : \chi_j = k\}$. Define the *utilization* of task j at level k as

$$u_j(k) = \frac{c_j(k)}{p_j}, \qquad j = 1, \ldots, n,\ k = 1, \ldots, \chi_j;$$

Define the total utilization at level k of jobs with criticality level i as

$$U_i(k) = \textstyle\sum_{j \in L_i} u_j(k), \qquad i = 1, \ldots, K,\ k = 1, \ldots, i.$$

It is well-known that in the case of a *single* criticality level, an (implicit-deadline) task system is feasible on m processors of speed σ if and only if $U_1(1) \leq \sigma m$ and $u_j(1) \leq \sigma$ for all j. The following necessary condition for MC-schedulability is an easy consequence of that fact.

Proposition 1. *If T is MC-schedulable on a unit speed processor, then for each $k = 1, \ldots, K$,*

$$\sum_{i=k}^{K} U_i(k) \leq 1.$$

In particular, when $K = 2$,

$$U_1(1) + U_2(1) \leq 1, \text{ and} \tag{1}$$

$$U_2(2) \leq 1. \tag{2}$$

Proof. For each k, consider the scenario where each task $j \in L_k \cup \cdots \cup L_K$ releases jobs with execution time $c_j(k)$. □

In the sequel we will call a job *active* if it has been released but has not yet been completed. In the case of a single criticality level and a single machine, it is known that the Earliest Deadline First algorithm, which schedules, preemptively, from among the active jobs the one with earliest deadline first, is optimal [8].

Proposition 2. *A set of tasks with a single criticality level is feasibly scheduled by the Earliest Deadline First algorithm on a single processor of speed σ if and only if $U_1(1) \leq \sigma$.*

3 Single Machine

The scheduling of a collection of independent mixed-criticality jobs on a single processor has already been treated in reference [2], where an algorithm with a speedup bound of 1.619 has been provided. Thus, in this section we focus on task systems. To facilitate understanding we start exposing the case of only 2 criticality levels and then we present the general result.

3.1 Two Criticality Levels

We consider a variant of the Earliest Deadline First algorithm that we call EDF with Virtual Deadlines (EDF-VD).

Algorithm EDF-VD. We distinguish two cases.

Case 1. $U_1(1) + U_2(2) \leq 1$. Apply EDF to the unmodified deadlines of the jobs. As soon as the system reaches level 2, that is a job executes for more than its WCET at level 1, cancel jobs from tasks in L_1.

Case 2. $U_1(1) + \frac{U_2(1)}{1-U_2(2)} \leq 1$ and Case 1 does not hold. Set $\lambda = \frac{U_2(1)}{1-U_1(1)}$. Then, while the system is in level 1: define for task $i \in L_2$, $\hat{p}_i = \lambda p_i$; redefine deadlines by adding $\hat{p}_i$ to the release time of each job of task $i \in L_2$, leaving the deadlines of jobs of tasks in L_1 as they were, and apply EDF to the modified deadlines. As soon as the system reaches level 2: cancel jobs from tasks in L_1; reset the deadlines of each job of task $i \in L_2$, by adding p_i to its release time; apply EDF to these (original) deadlines. □

We give the following sufficient condition for MC-schedulability by EDF-VD.

Theorem 1. *Assume T satisfies*

$$U_1(1) + \min\left(U_2(2), \frac{U_2(1)}{1-U_2(2)}\right) \leq 1.$$

Then T is schedulable by EDF-VD on a unit-speed processor.

Proof. If $U_1(1) + U_2(2) \leq 1$, it is clear that EDF-VD schedules the task system correctly. Therefore assume

$$U_1(1) + \frac{U_2(1)}{1-U_2(2)} \leq 1. \tag{3}$$

First we show that any level-1 scenario is scheduled correctly. The utilization of the task system with the modified deadlines is

$$\begin{aligned}
&\sum_{i\in L_1} \frac{c_i(1)}{p_i} + \sum_{i\in L_2} \frac{c_i(1)}{\lambda p_i} = \sum_{i\in L_1} \frac{c_i(1)}{p_i} + \sum_{i\in L_2} \frac{c_i(1)}{\frac{U_2(1)}{1-U_1(1)} p_i} \\
&= U_1(1) + \frac{1-U_1(1)}{U_2(1)} \sum_{i\in L_2} \frac{c_i(1)}{p_i} = U_1(1) + \frac{1-U_1(1)}{U_2(1)} U_2(1) = 1.
\end{aligned}$$

Thus, as long as the system is in level-1 it is scheduled correctly by Proposition 2, even satisfying the modified deadlines. Now we have to show that level-2 scenarios are scheduled correctly.

Let t^* denote the time-instant at which the system reaches level 2. Suppose that a job of level-2 task τ_j is released at t^* and is active (i.e., has been released but not yet completed execution). Let r_j denote its arrival time. The real deadline of this job – the time-instant by which it must complete execution – is $d_j := r_j +$

p_i; however, it is EDF-scheduled by EDF-VD assuming a deadline $\hat{d}_j := r_j + \hat{p}_i$. Since the job is active at t^*, it must be that $t^* \leq \hat{d}_j$, since, as we argued above, all jobs would meet their modified deadlines in the schedule for any level-1 scenario. Note that

$$d_j - \hat{d}_j = (r_j + p_i) - (r_j + \hat{p}_i) = p_i - \lambda p_i.$$

This implies that at time t^* for each level-2 job i at least $(1-\lambda)p_i$ time is left to finish the job in time. We will show that the artificial task system with only level-2 tasks (hence with only a single criticality level) having periods $(1-\lambda)p_i$ has total utilization less than 1, implying that this system is scheduled correctly by EDF, which implies that the original system can be scheduled correctly from time t^* onwards. From (3) we obtain

$$\frac{U_2(1)}{1-U_2(2)} \leq 1 - U_1(1) \Rightarrow \lambda = \frac{U_2(1)}{1-U_1(1)} \leq 1 - U_2(2) \Rightarrow 1 - \lambda \geq U_2(2)$$

Hence, the total utilization of the artificial task system is

$$\sum_{i \in L_2} \frac{c_i}{(1-\lambda)p_i} \leq \sum_{i \in L_2} \frac{c_i}{U_2(2)p_i} = 1.$$

□

The above schedulability condition can now be used to obtain a speedup guarantee. Let $\phi = (\sqrt{5}+1)/2 < 1.619$, the golden ratio.

Theorem 2. *If T satisfies*

$$\max(U_1(1) + U_2(1), U_2(2)) \leq 1,$$

then it is schedulable by EDF-VD on a speed ϕ processor. In particular, if T is MC-schedulable on a unit-speed processor, it is schedulable by EDF-VD on a speed ϕ processor (cf. Proposition 1).

Proof. We show the equivalent claim: if

$$\max(U_1(1) + U_2(1), U_2(2)) \leq 1/\phi, \tag{4}$$

then it is schedulable by EDF-VD on a speed 1 processor. Let $\Phi := 1/\phi$. We distinguish two cases.
Case $U_2(1) \geq \Phi U_1(1)$**.** By (4),

$$\Phi \geq U_1(1) + U_2(1) \geq (1+\Phi)U_1(1).$$

This gives

$$U_1(1) \leq \frac{\Phi}{1+\Phi} = \Phi^2.$$

So

$$U_1(1) + U_2(2) \leq \Phi^2 + \Phi = 1$$

and by Lemma 1, EDF-VD correctly schedules T on a processor of speed 1.
Case $U_2(1) \leq \Phi U_1(1)$ Again, by (4),

$$\Phi \geq U_1(1) + U_2(1) \geq \frac{1}{\Phi} U_2(1) + U_2(1) = \frac{\Phi + 1}{\Phi} U_2(1) = \frac{1}{\Phi^2} U_2(1).$$

Rewriting gives $U_2(1) \leq \Phi^3$. Then

$$U_1(1) + \frac{U_2(1)}{1 - U_2(2)} = U_1(1) + U_2(1) + U_2(1) \frac{U_2(2)}{1 - U_2(2)} \leq U_1(1) + U_2(1) + U_2(1) \frac{\Phi}{1 - \Phi}.$$

Since $1 - \Phi = \Phi^2$,

$$U_1(1) + \frac{U_2(1)}{1 - U_2(2)} \leq U_1(1) + U_2(1) + U_2(1) \frac{\Phi}{\Phi^2} \leq \Phi + \frac{\Phi^3}{\Phi} = 1$$

and by Lemma 1, EDF-VD correctly schedules T on a unit-speed processor. □

3.2 K Criticality Levels

As we have seen, the cases defining the EDF-VD algorithm are directed by the sufficient conditions of Theorem 1. Therefore before defining the algorithm for the K-level problem we first state the sufficient conditions for this case.

For the K-level problem there are $K - 1$ conditions, any of which being satisfied is a sufficient condition for schedulability by EDF-VD(K).

Theorem 3. *Assume T satisfies one of the following, for $k = 1, \ldots, K - 1$:*

$$\sum_{i=k}^{K-1} U_i(i) + \min \left\{ U_K(K), \frac{U_K(K-1)}{1 - \frac{U_K(K)}{\prod_{j=1}^{k}(1-\lambda_j)}} \right\} \leq \prod_{j=1}^{k} (1 - \lambda_j), \tag{5}$$

where $\lambda_1 = 0$ and, for $j > 1$,

$$\lambda_j = \frac{\sum_{\ell=j}^{K} U_\ell(j-1)}{\prod_{\ell=1}^{j-1}(1-\lambda_\ell)} \Big/ \left(1 - \frac{U_{j-1}(j-1)}{\prod_{\ell=1}^{j-1}(1-\lambda_\ell)} \right) \tag{6}$$

Then T is MC-schedulable on a unit-speed processor.

We define the algorithm by presenting the actions if condition k is satisfied for each $k = 1, \ldots, K - 1$. In the description of the algorithm we use next to the scaling parameters λ_ℓ defined in (6), scaling parameters μ_k defined by

$$\mu_k = \frac{U_K(K-1)}{\prod_{j=1}^{k}(1-\lambda_j)} \Big/ \left(1 - \frac{\sum_{i=k}^{K-1} U_i(i)}{\prod_{j=1}^{k}(1-\lambda_j)} \right).$$

Denote by $t^*_{\ell-1}$ the time instant where the system leaves level $\ell - 1$ and enters level ℓ. The periods of the tasks are modified in many levels. Once the system

switches to level ℓ, an active job of task τ_i can be seen as a task with a new (virtual) period, denoted by $p_i^{(\ell)}$, where $p_i^{(\ell)} = (1-\lambda_\ell)p_i^{(\ell-1)}$ and $p_i^{(1)} = p_i$.

Algorithm EDF-VD(K). Suppose that condition k of (5) holds and that conditions $1, \ldots, k-1$ do not hold.

Case 1. $U_K(K) \leq \frac{U_K(K-1)}{1-U_K(K)/\prod_{j=1}^{k}(1-\lambda_j)}$. Then, while the system is in level $\ell \leq k-1$: discard all jobs from tasks in $L_1, \ldots, L_{\ell-1}$; define for task $\tau_i \in L_j$, for $j = \ell+1, \ldots, K$, $\hat{p}_i^{(\ell)} = \lambda_{\ell+1} p_i^{(\ell)}$; for job J_h from task τ_i define a virtual release time $r_h^{(\ell)} = \min\{t_{\ell-1}^*, r_h\}$ and redefine deadlines $\hat{d}_h^{(\ell)} = r_h^{(\ell)} + \hat{p}_i^{(\ell)}$; apply EDF to the modified deadlines.
As soon as the system reaches level k, cancel jobs from tasks in $L_1, \ldots, L_{k-1}$ and reset the deadlines of each job of task $\tau_i \in L_j$, for $j = k, \ldots, K$, by adding p_i to its release time; apply EDF to these (original) deadlines.

Case 2. $U_K(K) > \frac{U_K(K-1)}{1-U_K(K)/\prod_{j=1}^{k}(1-\lambda_j)}$. Then, while the system is in level $\ell \leq k-1$: discard all jobs from tasks in $L_1, \ldots, L_{\ell-1}$; define for task $\tau_i \in L_j$, for $j = \ell+1, \ldots, K$, $\hat{p}_i^{(\ell)} = \lambda_{\ell+1} p_i^{(\ell)}$; for job J_h from task τ_i define a virtual release time $r_h^{(\ell)} = \min\{t_{\ell-1}^*, r_h\}$ and redefine deadlines $\hat{d}_h^{(\ell)} = r_h^{(\ell)} + \hat{p}_i^{(\ell)}$; apply EDF to the modified deadlines.
As soon as the system reaches level k: cancel jobs from tasks in $L_1, \ldots, L_{k-1}$; reset the deadlines of each job of task $\tau_i \in L_j$, for $j = k, \ldots, K-1$, by adding p_i to its release time; reset the deadlines of each job J_h from task $i \in L_K$ by adding $\mu_k p_i^{(k)}$ to its virtual release time $r_h^{(k)} = t_{k-1}^*$; apply EDF to these deadlines. □

4 Multiple Identical Machines

4.1 Scheduling a finite collection of independent jobs

For a single machine, Baruah et al. [2] analyzed the *Own Criticality Based Priority* (OCBP) rule and showed that it guarantees a speedup bound of ϕ on a collection of independent jobs. We show that this approach can be extended to multiple identical machines at the cost of a slightly increased bound.

The OCBP rule consists in determining, before knowing the actual execution times, a fixed priority ordering of the jobs and for each scenario execute at each moment in time the available job with the highest priority.

The priority list is constructed recursively. First the algorithm searches for a job that can be assigned the *lowest* priority: job J_k is a lowest priority job if there is at least $c_k(\chi_k)$ time in $[r_k, d_k]$ assuming that each other job J_j is executed before J_k for $c_j(\chi_k)$ units of time. This procedure is recursively applied to the collection of jobs obtained by excluding the lowest priority job J_k, until all the jobs are ordered or a lowest priority job cannot be found. A collection of jobs is called *OCBP-schedulable* if the algorithm finds a complete ordering of the jobs.

Algorithm 1 Own Criticality Based Priority (OCBP)

```
for i = 1 to n do
  if ∃ J_k ∈ J such that there is at least c_k(χ_k) time in [r_k, d_k] assuming each other
  job J_j is executed before J_k for c_j(χ_k) units of time then
    assign priority i to k (higher index denotes higher priority)
    J ← J \ {J_k}
  else
    return not OCBP-schedulable
  end if
end for
```

Theorem 4. *Let J be a collection of jobs that is schedulable on m unit-speed processors. Then J is schedulable using OCBP on m processors of speed $\phi + 1 - 1/m$.*

Proof. Let J be a minimal instance that is MC-schedulable on a processor and not OCBP-schedulable on m processors that are s times as fast for some $s > 1$.

Let $\gamma_1 = \sum_{j|\chi_j=1} c_j(1)$ denote the cumulative WCET for jobs with criticality level 1, and let $\gamma_2(1) = \sum_{j|\chi_j=2} c_j(1)$ and $\gamma_2(2) = \sum_{j|\chi_j=2} c_j(2)$ denote the WCETs for jobs with criticality level 2 at level 1 and 2, respectively. Let d_1 and d_2 denote the latest deadlines at level 1 and 2, respectively and let j_1 and j_2 denote the jobs with deadlines d_1 and d_2.

Lemma 1. *The job in J with latest deadline must be of criticality 2. This implies that $d_1 < d_2$.*

Proof. Suppose that $d_1 \geq d_2$. Consider the collection of jobs J' obtained from J by setting the level-2 WCET of criticality-2 jobs to their level-1 WCET. The MC-schedulability of J implies that J' is MC-schedulable. If j_1 cannot be a lowest priority job for J on s-speed processors, then there is not enough available execution time between the release time of j_1 and d_1 if all the other jobs J_j are executed for $c_j(1)$ units of time. Then, level-1 behaviors of J' cannot be scheduled, which is a contradiction. □

A work-conserving schedule is a schedule that never leaves a processor idle if there is a job available. For each $\ell \in \{1, 2\}$, we define Λ_ℓ as the set of time intervals on which all the processors are idle before d_ℓ for any work-conserving schedule and λ_ℓ as the the total length of this set of intervals.

Lemma 2. *For each $\ell \in \{1, 2\}$, and for each job J_j in J such that $\chi_j \leq \ell$, we have $[r_j, d_j] \cap \Lambda_\ell = \emptyset$. This implies that $\lambda_2 = 0$.*

Proof. Any job J_j in J such that $\chi_j \leq \ell$ and $[r_j, d_j] \cap \Lambda_\ell \neq \emptyset$ would meet its deadline if it were assigned lowest priority. As J is not OCBP-schedulable on a speed-s processor, then the collection of jobs obtained by removing such J_j from J is also not OCBP-schedulable. This contradicts the minimality of J. □

Since J is MC-schedulable on m speed-1 processors then γ_1 cannot exceed $(d_1 - \lambda_1) \cdot m$ in any criticality 1 scenario. Moreover, in scenarios where all jobs executes for their WCET at criticality 1, $\gamma_1 + \gamma_2(1)$ cannot exceed $(d_2 - \lambda_1) \cdot m$ and in scenarios where all jobs execute for their WCET at criticality 2, $\gamma_2(2)$ cannot exceed $(d_2 - \lambda_2) \cdot m$. Hence, the following inequalities hold:

$$\gamma_1 \leq (d_1 - \lambda_1)m \tag{7}$$
$$\gamma_1 + \gamma_2(1) \leq (d_2 - \lambda_1)m \leq d_2 \cdot m \tag{8}$$
$$\gamma_2(2) \leq (d_2 - \lambda_2)m = d_2 \cdot m. \tag{9}$$

Since J is not OCBP-schedulable on m speed-s processor, j_1 and j_2 cannot be the lowest priority jobs on such a processor. This implies that

$$\frac{1}{m}(\gamma_1 + \gamma_2(1) - c_{j_1}(1)) + c_{j_1}(1) > (d_1 - \lambda_1)s$$
$$\frac{1}{m}(\gamma_1 + \gamma_2(2) - c_{j_2}(2)) + c_{j_2}(2) > (d_2 - \lambda_2)s = sd_2$$

Hence:

$$\gamma_1 + \gamma_2(1) > sm(d_1 - \lambda_1) - (m-1)c_{j_1}(1) \tag{10}$$
$$\gamma_1 + \gamma_2(2) > smd_2 - (m-1)c_{j_2}(2). \tag{11}$$

Let us define $x = (d_1 - \lambda_1)/d_2$. By inequalities (8) and (10), it follows that

$$d_2 \cdot m > sm(d_1 - \lambda_1) - (m-1)c_{j_1}(1) \Rightarrow \left(1 - \frac{1}{m}\right) c_{j_1}(1) + d_2 > s(d_1 - \lambda_1).$$

By the MC-schedulability, we have: $c_{j_1}(1) \leq d_1 - \lambda_1$. Therefore

$$\left(1 - \frac{1}{m}\right)(d_1 - \lambda_1) + d_2 > s(d_1 - \lambda_1), \Rightarrow s < 1 - \frac{1}{m} + \frac{d_2}{d_1 - \lambda_1} = 1 + \frac{1}{x} - \frac{1}{m}.$$

By inequalities (7), (9), (11), we obtain

$$(d_1 - \lambda_1)m + d_2 \cdot m \geq \gamma_1 + \gamma_2(2) > smd_2 - (m-1)c_{j_2}(2).$$

Hence, $d_1 + d_2 + (1 - 1/m)c_{j_2}(2) > sd_2$, By the MC-schedulability, we have $c_{j_2}(2) \leq d_2$. Therefore,

$$d_1 - \lambda_1 + \left(2 - \frac{1}{m}\right) d_2 > sd_2, \Rightarrow s < \frac{d_1 - \lambda_1}{d_2} + 2 - \frac{1}{m} = x + 2 - \frac{1}{m}.$$

Hence,

$$s < \min\left\{1 + \frac{1}{x} - \frac{1}{m}, x + 2 - \frac{1}{m}\right\}.$$

As $x + 2 - \frac{1}{m}$ increases and $1 + \frac{1}{x} - \frac{1}{m}$ decreases, with increasing x, then the minimum value of s occurs when $x + 2 - \frac{1}{m} = 1 + \frac{1}{x} - \frac{1}{m}$, that is $x = \phi - 1$ and $s < 1 + \phi - \frac{1}{m}$. □

4.2 Scheduling a sporadic task system

We finally turn to the scheduling of a two-level mixed-criticality sporadic task system on multiple identical machines. We consider *partitioned* algorithms, that is, scheduling algorithms that partition the tasks on the machines and then schedule each machine independently (no migration). Using Theorem 2, the partitioning problem becomes a two-dimensional *vector scheduling* problem [5] where the vectors to be packed are the utilization vectors (u_j) of the tasks. The two-dimensional vector scheduling problem can be approximated in polynomial time within a factor $1+\epsilon$ for any $\epsilon > 0$ [5], and we are able to derive the following theorem.

Theorem 5. *For any $\epsilon > 0$ there is a polynomial-time partitioning algorithm P such that any task system that is MC-schedulable by some partitioned algorithm on m unit-speed processors can be scheduled by the combination of P and EDF-VD on m speed $\phi + \epsilon$ processors.*

References

1. Barhorst, J., Belote, T., Binns, P., Hoffman, J., Paunicka, J., Sarathy, P., Stanfill, J.S.P., Stuart, D., Urzi, R.: White paper: A research agenda for mixed-criticality systems (2009), available at `http://www.cse.wustl.edu/~cdgill/CPSWEEK09_MCAR/`
2. Baruah, S.K., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., Stougie, L.: Scheduling real-time mixed-criticality jobs. In: Hliněný, P., Kučera, A. (eds.) Proc. 35th Symp. on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, vol. 6281, pp. 90–101. Springer (2010)
3. Baruah, S.K., Goossens, J.: Scheduling real-time tasks: Algorithms and complexity. In: Leung, J.Y.T. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chap. 28. CRC Press (2003)
4. Baruah, S.K., Li, H., Stougie, L.: Towards the design of certifiable mixed-criticality systems. In: Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium. pp. 13–22. IEEE (2010)
5. Chekuri, C., Khanna, S.: On multidimensional packing problems. SIAM Journal on Computing 33(4), 837–851 (2004)
6. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. Journal of the ACM 47(4), 617–643 (2000)
7. Li, H., Baruah, S.K.: An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In: Proc. 16th IEEE Real-Time Systems Symp. pp. 183–192. IEEE (2010)
8. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1), 46–61 (1973)