Load-based Schedulability Analysis of Certifiable Mixed-criticality Systems

Haohan Li
Department of Computer Science
The University of North Carolina
Chapel Hill, NC. USA
lihaohan@cs.unc.edu

Sanjoy Baruah
Department of Computer Science
The University of North Carolina
Chapel Hill, NC. USA
baruah@cs.unc.edu

ABSTRACT

Many safety-critical embedded systems are subject to certification requirements. However, only a subset of the functionality of the system may be safety-critical and hence subject to certification; the rest of the functionality is non safety-critical and does not need to be certified. Certification requirements in such mixed-criticality systems give rise to some interesting scheduling problems, that cannot be satisfactorily addressed using techniques from conventional scheduling theory. In prior work, we have proposed a priority-based algorithm for scheduling such mixed-criticality systems on preemptive uniprocessor platforms. In this paper, we derive a sufficient schedulability condition for efficiently determining whether a given mixed-criticality system can be successfully scheduled by this algorithm. We show that this algorithm (and the associated schedulability test) is strictly superior to prior algorithms that have been used for scheduling mixed-criticality systems needing certification.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*; G.4 [**Mathematical software**]: Certification and testing

General Terms

Verification

Keywords

Preemptive uniprocessor scheduling; certification; priority-based scheduling; load bounds.

1. INTRODUCTION

Many embedded systems perform safety-critical functions that must be certified correct by statutory organizations. However, the current trend towards integrating multiple functionalities on a common platform, driven by cost and "SWaP" (*Size*, *Weight*, and *Power*– i.e., energy– consumption) considerations means that even in highly safety-critical systems, typically only a relatively small fraction of the overall system is actually of critical functionality and hence needs to be certified. The remainder of the system is comprised of non-critical code that enhances the overall performance of the system, but does not effect safety and is therefore not subject to certification. Such *mixed criticality* systems are becoming increasingly common in embedded systems; in fact, mixed criticalities have been identified as among the core foundational concepts in the emerging discipline of Cyber Physical Systems. Coming up with procedures that will allow for the cost-effective certification of such mixed-criticality systems has been identified as a unique, particularly challenging, collection of problems [4]. Recognizing these challenges, several US government R&D organizations including AFRL, NSF, NSA, NASA, etc., have led initiatives such as the Mixed Criticality Architecture Requirements (MCAR) program aimed at streamlining the certification process for safety-critical embedded systems; these initiatives have brought together participants from industry, academia, and standards bodies to seek out more advanced, efficient, and cost-effective certification processes. *It is this aspect of mixed criticalities arising as a consequence of such certification requirements, that is the focus of the research described in this paper*.

In order to certify a system as being correct, the certification authority (CA) must make certain assumptions about the worst-case behavior of the system during run-time. In this paper, we focus on one particular aspect of run-time behavior: the *worst-case execution time (WCET)* of pieces of code. CA's tend to be very conservative, and hence it is often the case that the WCET estimates used by the CA will be far more pessimistic than those the system designer would typically use during the system design process. On the other hand, while the CA is only concerned with the correctness of the safety-critical part of the system the system designer wishes to ensure that the entire system is correct, including the non-critical parts. We illustrate by a contrived example.

Example 1. Consider a system to be implemented on a preemptive uniprocessor, that is comprised of three jobs J_1, J_2, and J_3. All three jobs are released at time zero. Job J_1 has a deadline at time-instant 2, while the other two jobs have their deadlines at time-instant 3.5. Jobs J_2 and J_3 are high-criticality and subject to certification, whereas J_1 is low-criticality and hence not.

- The system designer is confident that each job has a

WCET not exceeding 1. Hence executing the jobs in earliest deadline first (EDF) order will ensure that all three complete by their deadlines.

- However, the CA chooses to use more pessimistic WCET estimates during the certification process, and claims that jobs J_2 and J_3 may each need 1.5 time units of execution[1].

If the system were indeed scheduled using EDF, the CA would determine that in the worst case, J_1 executes over $[0,1)$ and the job from among J_2 and J_3 that is next chosen for execution will execute for 1.5 time units, thereby causing the other high-criticality job to miss its deadline at time 3.5. *The system scheduled using EDF would therefore fail certification.*

On the other hand if we were to assign greater priority to the high-criticality jobs, then they would both meet their deadlines even under the worst-case scenarios envisioned by the CA. *However the low-criticality job J_1 will miss its deadline even when each job executes for at most 1 time unit (as predicted by the system designer).*

It turns out that a "correct" scheduling strategy[2] for this system is as follows:

- Execute J_2 over $[0,1)$.
- If J_2 completes execution at time-instant 1, then execute J_1 over $[1,2)$ and J_3 over $[2,3.5)$, thereby ensuring that all deadlines are met.
- If J_2 does not complete execution by time-instant 1, then discard J_1 and continue the execution of J_2, following that with the execution of J_3 over $[1.5,3)$. Both the high-criticality jobs will complete by their deadlines in the worst-case scenario envisioned by the CA.

Under this scheduling strategy, it may be verified that the system both passes certification, *and* meets all deadlines when it behaves as expected to by the system designer. □

This research.

The central thesis of this research is that the efficient utilization of computing resources in mixed-criticality systems that are subject to certification requirements requires the development of new scheduling theory. The insight we seek to exploit in developing such theory is this. Certification is performed under conservative assumptions: the CA makes very pessimistic assumptions about the run-time behavior of the system, and requires that it be demonstrated correct under these pessimistic assumptions. In order to perform system certification under such pessimism one must, informally speaking, severely *over-provision* computing resources to the part of the system needing certification. Some of this over-provisioned capacity could then be *reclaimed*, during system design and analysis time itself, to make performance guarantees to the remainder of the system, since such guarantees are made under a correspondingly lower degree of pessimism.

[1]The CA may also determine that the low-criticality job J_1 needs more than 1 time unit to complete execution; however, let us assume for now that the system is implemented to abort the execution of J_1 if it "times out" and executes for more than 1 time-unit.

[2]The notion of a correct scheduling strategy is formally defined in Section 2.

Organization of this paper.

In Section 2, we present the formal model for representing mixed-criticality real-time systems that is used in this research. This formal model extends the conventional model of a real-time job by allowing for the specification of two different WCET's, one at each criticality level, for a single job. In Section 3 we briefly survey some other work on mixed-criticality real-time systems, focusing in particular on the results that we will be using in later sections. In Section 4 we derive our new sufficient schedulability condition, and prove that it is strictly more general than a previously proposed schedulability condition. In Section 5 we demonstrate that the results derived here are superior to some other widely-used approaches towards certification-cognizant scheduling of mixed-criticality systems.

2. MODEL AND DEFINITIONS

In this section we formally define the mixed-criticality job model that is used in this paper, and explain terms and concepts used throughout the remainder of this document.

Although our eventual interest is in the scheduling of collections of recurrent mixed-criticality task systems potentially comprised of an infinite number of jobs[3], many fundamental questions remain unanswered regarding even the simpler case of finite collections of jobs. Hence we focus in this paper on the simpler case where a system is comprised of a finite number of jobs. Our results may be considered as a step towards a more comprehensive analysis of systems of mixed-criticality recurrent tasks. In addition, these results have immediate applicability for the scheduling of *frame-based* recurrent real-time systems, in which the recurrent nature of the behavior is expressed as the infinite repetition of a finite collection of jobs of the kind considered here.

For the purposes of this paper, a mixed-criticality (MC) *job* is characterized by a tuple of five parameters: $J_i = (A_i, D_i, \chi_i, C_i, C'_i)$, where

- $A_i \in R^+$ is the release time.
- $D_i \in R^+$ is the deadline. We assume that $D_i \geq A_i$.
- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the job. A HI-criticality job (a J_i with $\chi_i = \text{HI}$) is one that is subject to certification, whereas a LO-criticality job (a J_i with $\chi_i = \text{LO}$) is one that does not need to be certified.
- C_i specifies the worst case execution time (WCET) estimate of J_i that is used by the system designer (i.e., the WCET estimate at the LO criticality level).
- C'_i specifies the worst case execution time (WCET) estimate of J_i that is used by the certification authorities (i.e., the WCET estimate at the HI criticality level).We assume that
  - $C'_i \geq C_i$ (i.e., the WCET estimate used by the system designer is never more pessimistic than the one used by the CA), and
  - $C'_i = C_i$ if $\chi_i = \text{LO}$ (i.e., a LO-criticality job is aborted if it executes for more than its LO-criticality WCET estimate).

[3]Some partial results concerning the scheduling of such task systems may be found in [14, 8].

Where do these values C_i and C'_i come from, and why are they different? It is well known that determining exact worst-case execution times of pieces of code is a very difficult problem; instead, systems engineers work with *upper bounds* on the exact value. However, for many non-trivial kinds of code strict upper bounds are extremely pessimistic, and represent scenarios that are highly unlikely to occur in practice. For such code, less pessimistic upper bounds on their WCET's may be obtained at lower degrees of confidence than absolute certainty. Based on the observation that "the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice," Vestal [14] proposed that multiple different WCET values be specified, with the different values being determined at a different level of assurance. These different values may be obtained by using different execution-time analysis tools; we expect that the tool used by the CA is more conservative than the one used by the system engineer, and hence the CA's WCET estimates (the C'_i values) are larger than the estimates used during the design process (the C_i values).

§2.1. MC instance.

An MC instance is specified as a finite collection of such MC jobs: $I = \{J_1, J_2, \ldots, J_n\}$. Given such an instance, we are concerned here with determining how to schedule it to obtain correct behavior; in this document, we restrict our attention to scheduling on preemptive uniprocessor platforms.

§2.2. Loads ℓ_{LO} and ℓ_{HI}.

In classical real-time scheduling theory (see, e.g.,[12, page 81]), the *load* of an instance denotes the maximum over all time intervals, of the cumulative execution requirement by jobs of the instance over the interval, normalized by the interval length. Informally, the load of an instance represents a lower bound on the speed of any processor upon which it can meet all deadlines.

Analogous to this concept, we find it convenient to define two loads, $\ell_{\text{LO}}(I)$ and $\ell_{\text{HI}}(I)$, of a MC instance I:

Definition 1. *The* LO*-criticality load $\ell_{\text{LO}}(I)$ and the* HI*-criticality load $\ell_{\text{HI}}(I)$ of a mixed-criticality instance I are defined according to the following two formulas:*

$$\ell_{\text{LO}}(I) = \max_{0 \le t_1 < t_2} \frac{\sum_{J_i \,:\, t_1 \le A_i \wedge D_i \le t_2} C_i}{t_2 - t_1}$$

$$\ell_{\text{HI}}(I) = \max_{0 \le t_1 < t_2} \frac{\sum_{J_i \,:\, \chi_i = \text{HI} \wedge t_1 \le A_i \wedge D_i \le t_2} C'_i}{t_2 - t_1}$$

□

Informally, $\ell_{\text{LO}}(I)$ is the largest load that the system designer expects to have to deal with during run-time while executing instance I, whereas $\ell_{\text{HI}}(I)$ denotes the load of the part of I that the CA seeks to certify. Clearly, it is necessary (albeit not sufficient) that both $\ell_{\text{LO}}(I)$ and $\ell_{\text{HI}}(I)$ be no larger than the speed of the processor on which I is to be executed, if all deadlines are to be met (from the designer's perspective) and the system is to be certified correct.

Both $\ell_{\text{LO}}(I)$ and $\ell_{\text{HI}}(I)$ for an MC instance I with n jobs can be determined in time that is polynomial in n. To see this, we observe that only such values of t_1 and t_2 need be considered where t_1 is equal to some A_i and t_2 is equal to some D_i. There are no more than n^2 possible such $[t_1, t_2)$ intervals, and computing the sum of the WCET estimates over each interval takes $O(n)$ time. Thus even with the brute-force method, we can compute both loads in $O(n^3)$ time.

§2.3. Behaviors.

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.

Definition 2 (the criticality level of a behavior). *The* criticality level *of the behavior $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ of I is defined to be* LO *if $\gamma_i \le C_i$ for all $i, 1 \le i \le n$, and* HI *if it is not* LO *but $\gamma_i \le C'_i$ for all i. (If $\gamma_i > C'_i$ for any i, then we define that behavior to be* erroneous.*)* □

Informally speaking, the system designer fully expects that all behaviors will be at criticality level LO, and would like all jobs to complete by their deadlines. The CA, on the other hand, allows for the possibility that some behaviors may be of criticality level HI, and requires that all HI-criticality jobs meet their deadlines in the event of such behavior.

§2.4. Scheduling strategies.

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 becomes known by executing J_i until it signals that it has completed execution. Since these actual execution times – the γ_i's – only become revealed during run-time, an on-line scheduling strategy does not a priori know what the criticality level of any particular behavior is going to be; at each instant, scheduling decisions are made based only on the partial information revealed thus far.

§2.5. Correctness.

A scheduling strategy is said to be a *correct MC scheduling strategy* if it satisfies the following criteria:

- when scheduling any behavior of criticality level LO it ensures that every job J_i receives sufficient execution during the interval $[A_i, D_i)$ to signal that it has completed execution.
- when scheduling any behavior of criticality level HI, it ensures that every job J_i with χ_i = HI receives sufficient execution during the interval $[A_i, D_i)$ to signal that it has completed execution.

Informally, a correct scheduling strategy is one that "satisfies" both the system designer and the CA under their respective, different, assumptions: the designer believes that all deadlines will be met, while the CA can verify that all HI-criticality jobs will meet their deadlines.

§2.6. MC schedulability.

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 is to determine whether a given MC instance is MC schedulable or not.

It has been shown [5] that determining whether a given MC instance is MC schedulable or not is highly intractable:

THEOREM 1 (FROM [5]). *The MC schedulability problem — given an MC instance, determine whether it is MC-schedulable — is NP-hard in the strong sense. This hardness result holds even in the restricted case where all jobs in the MC instance have the same arrival times.*

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. (Of course, in practice we are interested not merely in determining whether a given instance is MC schedulable or not, but in actually constructing a correct scheduling strategy in the event that it is MC schedulable.)

3. RELATED WORK

As we have stated in Section 1, the strategic significance of mixed criticality certification is widely recognized and has been the subject of multiple workshops and working-group meetings, some of the findings of which are highlighted in a white paper [4]. Several interesting and innovative approaches for addressing this issue have recently been proposed. While providing a complete and comprehensive survey of all this research is beyond the scope of this document, we would like to highlight a few specific works that we consider particularly important and innovative.

To our knowledge, the scheduling problem that arises from multiple certification requirements, at different criticality levels, was first identified and formalized by Vestal in [14], in the context of the fixed-priority preemptive uniprocessor scheduling of recurrent task systems.

Current practice in safety-critical embedded systems design for certifiability is centered around the technique of "space-time partitioning," as codified in, e.g., the ARINC-653 standard [1, 15]. Loosely speaking, *space partitioning* means that each application is granted exclusive access to some of the physical resources on board the platform, and *time partitioning* means that the time-line is divided into slots with each slot being granted exclusively to some (pre-specified) application. Interactions among the partitioned applications may only occur through a severely limited collection of carefully-designed library routines.

This ARINC-653 approach is one of several *reservation-based* approaches, in which a certain amount of the capacity of the shared platform is reserved for each application, that have been considered for designing certifiable mixed-criticality systems. Although this approach works, as is evidenced by the fact that large, complex safety-critical embedded systems have been designed, built, and certified, and are currently widely deployed, it is known that reservation-based approaches tend to be pessimistic (in the sense of under-utilizing platform resource). This is a consequence of the very principle of isolation between criticality levels upon which reservations-based design techniques are based: such isolation rules out the possibility of reusing the resource capacity that must be assigned to high-criticality applications in order that they pass certification, but which they are unlikely to need in practice, to make performance guarantees to low-criticality applications. In Section 5 we explore more precisely the relationship between space-time partitioning and the approach to mixed-criticality scheduling that is advocated in this paper.

Priority-based scheduling is the other technique commonly used by systems engineers in dealing with mixed criticalities. Unless carefully designed, such priority-based scheduling schemes can be even more pessimistic than reservations-based approaches. In a typical priority-based scheduling approach, for example, jobs belonging to higher-criticality applications are accorded greater priority in deciding which job to execute at each instant in time. It is not too difficult to construct simple examples in which such *criticality-monotonic* scheduling will perform arbitrarily poorly. For instance, consider a 2-job system in which the LO-criticality job has a much earlier deadline, and a far smaller WCET, than the HI-criticality job. Although a criticality-monotonic schedule would pass certification in the sense that the HI-criticality job would complete by its deadline, the LO-criticality job would tend to miss its deadline during LO-criticality behaviors, even though there may be far more than adequate computing capacity to meet both jobs' WCET's at their specified criticality levels. These and other drawbacks of such a criticality-monotonic approach are highlighted in [9].

However, not all priority-based scheduling algorithms perform quite so poorly; in a recently published paper [7], a priority-based algorithm called OCBP has been proposed for mixed-criticality scheduling, that does not suffer from these shortcomings of criticality-monotonic scheduling. A quantitative evaluation of the effectiveness of OCBP was derived in [7, Lemma 5] in terms of its *processor speedup factor*; this result can be stated as follows. Given any instance that is MC-schedulable upon a unit-speed processor (by Theorem 1, determining this fact is NP-hard in the strong sense), OCBP successfully schedules it in polynomial time, upon a processor of speed $(\sqrt{5}+1)/2$ (i.e., ≈ 1.618). In other words, *a processor speedup factor of approximately* 1.618 *is sufficient for OCBP to deal with the intractability (NP-hardness in the strong sense) of MC-schedulability*. In subsequent work, we have [6] generalized this processor speedup result to MC systems with more than two criticality levels, deriving bounds on the maximum processor speedup necessary in order for (a generalization of) OCBP to schedule, in polynomial time, MC instances that may be subject to multiple different certification requirements.

Processor speedup factors are a useful conceptual characterization of the effectiveness of a scheduling algorithm, and may provide valuable insight into the algorithm's properties. However, it is not directly obvious how these processor speedup factors should be used by systems designers during the process of designing and implementing systems. In this paper, we therefore extend the work in [7] in a different direction, by seeking an alternate characterization of the schedulability properties of OCBP. This characterization is in terms of the parameters of the instances I that OCBP is able to schedule (specifically, the LO-criticality load $\ell_{\text{LO}}(I)$ and the HI-criticality load $\ell_{\text{HI}}(I)$). Since, as explained in

Section 2 above, these parameters are easily and efficiently computed for any instance I, such a characterization of the schedulability properties of OCBP makes it easy to determine whether a given instance is schedulable or not using OCBP.

Some other research on mixed-criticality scheduling. Although many other real-time scheduling papers deal with mixed-criticality systems, they do not really deal with scheduling for certification. De Niz et al. [9] deal with a different aspect of mixed-criticality systems from the one we focus on here, in that they do not directly address the certification issue. Nevertheless, [9] contains very interesting and novel ideas that merit mention. This work observes that the complete inter-criticality isolation offered by the reservations approach may cause *criticality inversion*: preventing a higher-criticality job from meeting its deadline while allowing lower-criticality jobs to complete. On the other hand, assigning priorities according to criticality may result in very poor processor utilization. An innovative *slack-aware* approach is proposed that builds atop priority-based scheduling (with priorities not necessarily assigned according to criticality), to allow for asymmetric protection of reservations thereby helping to lessen criticality inversion while retaining reasonable resource utilization.

Pellizzoni et al. [13], use a reservations-based approach to ensure strong isolation among sub-systems of different criticalities; this paper proposes innovative design and architectural techniques for preserving such isolation despite some necessary interaction (e.g., in the sharing of additional non-preemptable resources) between jobs of different criticalities. The focus is not on optimizing resource utilization, but on ensuring isolation; hence, this research does not attempt to avoid the criticality-inversion that is inherent to the reservations-based approach.

3.1 The OCBP scheduling algorithm

In prior work [7], we have derived a priority-based algorithm called ***OCBP*** (*Own Criticality-Based Priorities*) for mixed-criticality scheduling. The high-level description of the OCBP algorithm is as follows. Given a dual-criticality instance I, we determine off-line (i.e., prior to run-time) a total priority ordering of the jobs of I such that scheduling the jobs according to this priority ordering guarantees a correct schedule, where *scheduling according to a priority ordering* means that at each moment in time the highest-priority available job is executed.

The priority ordering is constructed recursively using the approach commonly referred to in the real-time scheduling literature as the "Audsley approach" [2, 3]. We first determine the lowest priority job: Job J_i may be assigned the lowest priority if

- it is a LO-criticality job ($\chi_i = \text{LO}$), and there is at least C_i time between its release time and its deadline available if every other job J_j has higher priority and is executed for C_j time units; or
- it is a HI-criticality job ($\chi_i = \text{HI}$), and there is at least C'_i time between its release time and its deadline available if every other job J_j has higher priority and is executed for C'_j time units[4].

[4]Recall that we assume that $C'_j = C_j$ for every J_j with $\chi_j = \text{LO}$; i.e., no LO-criticality job is permitted to execute for more than its LO-criticality WCET.

The above procedure is then repeated to the set of jobs excluding the lowest priority job, until all jobs are ordered, or at some iteration a lowest priority job does not exist. (If this happens, the priority-assignment algorithm reports failure and we say that the instance is not OCBP-schedulable.) We illustrate the operation of the OCBP priority assignment algorithm by an example:

EXAMPLE 2. Consider the instance comprised of the following three jobs. J_1 is not subject to certification, whereas J_2 and J_3 must be certified correct.

J_i	A_i	D_i	χ_i	C_i	C'_i
J_1	0	4	LO	2	2
J_2	0	5	HI	2	4
J_3	0	10	HI	2	4

Let us determine which, if any, of these jobs could be assigned lowest priority according to the OCBP priority assignment algorithm:

- If J_1 were assigned lowest priority, J_2 and J_3 could consume $C_1+C_2 = 2+2 = 4$ units of processor capacity over $[0, 4)$, thus leaving no execution for J_1 prior to its deadline.
- If J_2 were assigned lowest priority, J_1 and J_3 could consume $C'_1+C'_3 = 2+4 = 6$ units of processor capacity over $[0, 6)$, thus leaving no execution for J_2 prior to its deadline at time-instant 5.
- If J_3 were assigned lowest priority, J_1 and J_2 could consume $C'_1+C'_2 = 2+4 = 6$ units of processor capacity over $[0, 6)$. This leaves 4 units of execution for J_3 prior to its deadline at time-instant 10, which is sufficient for J_3 to execute for $C'_3 = 4$ time units. *Job J_3 may therefore be assigned the lowest priority.*

Next, the OCBP priority assignment algorithm would consider the instance $\{J_1, J_2\}$, and seek to assign one of these jobs the lower priority:

- If J_1 were assigned lower priority, J_2 could consume $C_2 = 2$ units of processor capacity over $[0, 2)$. This leaves 2 units of execution for J_1 prior to its deadline at time-instant 4, which is sufficient for J_1 to execute for $C_1 = 2$ time units. *Job J_1 may therefore be assigned the lowest priority from among $\{J_1, J_2\}$.*
- It may be verified that J_2 *cannot* be assigned the lowest priority from among $\{J_1, J_2\}$. If we were to do so, then J_1 could consume $C'_1 = 2$ units of processor capacity over $[0, 2)$. This leaves 3 units of execution for J_1 prior to its deadline at time-instant 5, which is not sufficient for J_2 to execute for the $C'_2 = 4$ time units it needs to complete on time.

The final OCBP priority ordering is therefore as follows. Job J_2 has the greatest priority, job J_1 has the next-highest priority, and J_3 has the lowest priority. It may be verified that scheduling according to these priorities is a correct MC scheduling strategy for the instance $\{J_1, J_2, J_3\}$, (recall from Section 2 the definition of "correct" scheduling strategies). □

The following properties of OCBP were proved in [7] (or follow directly from results that were proved there):

1. The OCBP priority assignment algorithm has a run time that is polynomial in the representation of the instance being scheduled.

2. If the OCBP priority assignment algorithm succeeds in assigning priorities to the jobs of an instance I, then priority-based scheduling of I according to these priorities is a correct MC scheduling strategy.

3. The OCBP priority assignment algorithm succeeds in assigning priorities to the jobs of any instance I that satisfies

$$\ell_{\text{LO}}(I) \leq \frac{\sqrt{5}-1}{2} \quad \text{and} \quad \ell_{\text{HI}}(I) \leq \frac{\sqrt{5}-1}{2}$$

(Recall that $(\sqrt{5}-1)/2 \approx 0.62$ is the famous mathematical constant commonly called the"Golden ratio", and often denoted Φ.)

4. LOAD-BASED SCHEDULABILITY ANALYSIS

As stated above, results from [7] can be used to conclude that any instance I for which both $\ell_{\text{LO}}(I)$ and $\ell_{\text{HI}}(I)$ are bounded from above at about 0.62, is guaranteed to be successfully scheduled by OCBP on a unit-speed processor. In this section, we consider systems in which these conditions are *not* satisfied: one of the two loads is larger than 0.62, while the other is smaller than this bound. This is a reasonable case to consider: it is unlikely that the LO-criticality and the HI-criticality loads of an instance will both be similarly constrained. We instead expect that $\ell_{\text{HI}}(I)$ would usually be rather small compared to $\ell_{\text{LO}}(I)$, since it is typically the case that a relatively small fraction of the code on board an integrated embedded platform is devoted to safety-critical (and hence certifiable) functionality. We would like to be able to determine the schedulability by OCBP of such instances. This is done in Theorem 2 below, which shows that $\ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) \leq 1$ is sufficient to ensure that I is successfully scheduled by OCBP. (Note that for "regular" – i.e., non MC – instances, this condition reduces to the trivial result that load ≤ 1 is sufficient for schedulability, since a non-MC instance can be considered to be a MC instance with one of $\ell_{\text{LO}}(I)$ or $\ell_{\text{HI}}(I)$ set equal to zero.)

THEOREM 2. *The OCBP priority assignment algorithm generates a priority ordering that yields a correct MC scheduling strategy for any MC instance I satisfying*

$$\ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) \leq 1 \ . \tag{1}$$

Proof: Let I denote a *minimal* instance for which the OCBP priority assignment fails to generate a priority ordering. We will prove that it must be the case that $\ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) > 1$. The theorem follows, by contradiction.

Without loss of generality, let us assume that $\min_{J_i \in I} A_i = 0$ (i.e., the earliest release time is zero).

Observe that it must be the case that there is no time-instant $t \in [0, \max_{J_i \in I} D_i)$ such that no job's scheduling window[5] contains t. If there were such a t, it would follow that either the instance comprised of only those jobs with scheduling windows before t, or the instance comprised of only those jobs with scheduling windows after t, is not OCBP-schedulable; this contradicts the assumed minimality of I.

[5]The *scheduling window* of a job J_i is the interval $[A_i, D_i)$.

OBSERVATION 2.1. *If the job[s] in I with the latest deadline are not [all] HI-criticality jobs, then Equation 1 is violated by I.*

Proof: If a LO-criticality job has the latest deadline but nevertheless cannot therefore be assigned lowest priority, it follows from the optimality of EDF for scheduling "regular" (non-MC) real-time workloads that the LO-criticality behavior of I in which each job J_i executes for C_i time units is not schedulable. This requires that $\ell_{\text{LO}}(I) > 1$, which in turns implies that Equation 1 is violated. ■

We have thus proved the theorem for those instances I in which the latest-deadline jobs are not all HI-criticality jobs. In the remainder of this proof we will consider the remaining case, when all the latest-deadline jobs in I are HI-criticality jobs. Let j_2 denote such a latest-deadline job with deadline d_2, and let j_1 denote the LO-criticality job with the latest deadline, this deadline being at $d_1 (d_1 < d_2)$.

Some additional notation: let c_{LO} denote the total LO-criticality WCET of all LO-criticality jobs in I, and let $c_{\text{HI}}(\text{LO})$ and $c_{\text{HI}}(\text{HI})$ respectively denote the total LO-criticality and the total HI-criticality WCET's of all the HI-criticality jobs:

$$\begin{aligned} c_{\text{LO}} &= \sum_{J_i \in I \mid \chi_i = \text{LO}} C_i \\ c_{\text{HI}}(\text{LO}) &= \sum_{J_i \in I \mid \chi_i = \text{HI}} C_i \\ c_{\text{HI}}(\text{HI}) &= \sum_{J_i \in I \mid \chi_i = \text{HI}} C_i' \end{aligned}$$

Consider now any work-conserving schedule of I where each job J_i requests exactly C_i time units of execution. Let $\Lambda_1, \Lambda_2, \cdots$ denote the intervals, during which the processor is idle in this schedule. We define their cumulative length as λ.

Since we're assuming that I is not OCBP-schedulable, it must be the case that j_1 cannot be the lowest-priority job on such a processor. Hence, it is necessary that

$$c_{\text{LO}} + c_{\text{HI}}(\text{LO}) > (d_1 - \lambda) \tag{2}$$

OBSERVATION 2.2. *For each $j \geq 1$, no J_i with criticality $\chi_i = \text{LO}$ has a scheduling window that overlaps with Λ_j.*

Proof: Suppose that some LO-criticality job J_i were to overlap with Λ_j. This means that in a LO-criticality behavior, all the jobs which arrive prior to Λ_j complete by the beginning of Λ_j. Hence, J_i would complete by its deadline in any behavior of criticality level one, if it were assigned lowest priority. But this contradicts the assumed non-OCBP-schedulability of I. ■

Since we assume that j_1 is the latest-deadline LO-criticality job, we know that every LO-criticality job has a deadline earlier than d_1. Thus by Observation 2.2 above and the definition of $\ell_{\text{LO}}(I)$ we get

$$c_{\text{LO}} \leq \ell_{\text{LO}}(I) \times (d_1 - \lambda) \tag{3}$$

This follows from the observation that in every time interval $[t_1, t_2)$ that overlaps with a LO-criticality job's scheduling window , we know that the summation of time demand over $[t_1, t_2)$ is no greater than $\ell_{\text{LO}}(I) \times (t_2 - t_1)$. Therefore the accumulation of time demand over all possible intervals will be no greater than $\ell_{\text{LO}}(I) \times (d_1 - \lambda)$.

From the definition of ℓ_{LO} and ℓ_{HI},we get the following two inequalities:

$$\begin{aligned} c_{\text{LO}} + c_{\text{HI}}(\text{LO}) &\leq \ell_{\text{LO}}(I) \times d_2 & (4) \\ c_{\text{HI}}(\text{HI}) &\leq \ell_{\text{HI}}(I) \times d_2 & (5) \end{aligned}$$

OBSERVATION 2.3. *Consider any work-conserving schedule of I, when each* LO*-criticality job J_i requests exactly C_i units of execution, and each* HI*-criticality job j_j requests exactly C'_j units of execution*[6]*. There are no idle intervals in this schedule.*

Proof: If there were an idle interval, any job whose scheduling window spans the idle interval would meet its deadline if it were assigned lowest priority. But this contradicts the assumed non-OCBP-schedulability of I. ■

Since we are assuming that I is not OCBP-schedulable, it must be the case that j_2 cannot be the lowest-priority job. Given Observation 2.3 above, it must then be the case that

$$c_{\text{LO}} + c_{\text{HI}}(\text{HI}) > d_2 \qquad (6)$$

From Inequalities 2 and 4, we can get

$$\ell_{\text{LO}}(I)\, d_2 > d_1 - \lambda \qquad (7)$$

We can simplify Inequality 6 above:

$$\begin{aligned} & c_{\text{LO}} + c_{\text{HI}}(\text{HI}) > d_2 \\ \Rightarrow\ & \text{From Inequalities 3, 5} \\ & \ell_{\text{LO}}(I)\ (d_1 - \lambda) + \ell_{\text{HI}}(I)\ d_2 > d_2 \\ \Rightarrow\ & \text{By Inequality 7} \\ & \ell_{\text{LO}}(I)\ (\ell_{\text{LO}}(I)\ d_2) + \ell_{\text{HI}}(I)\ d_2 > d_2 \\ \equiv\ & \ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) > 1 \end{aligned}$$

We have just proved that any minimal instance I for which the OCBP priority assignment algorithm fails to generate s priority ordering must have $\ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) > 1$. Hence OCBP successfully assigns priorities to any instance I satisfying $\ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) \leq 1$, and our theorem is proved. ■

Discussion.

We had pointed out in Section 3.1 that results in [7] allow us to conclude that any instance I, for which both $\ell_{\text{LO}}(I)$ and $\ell_{\text{HI}}(I)$ are no larger than $(\sqrt{5}-1)/2$, is successfully scheduled by OCBP. We will now show that this follows from Theorem 2. To see this, observe that for a system in which both $\ell_{\text{LO}}(I)$ and $\ell_{\text{HI}}(I)$ are no larger than $(\sqrt{5}-1)/2$,

$$\begin{aligned} & \ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) \\ \leq\ & (\frac{\sqrt{5}-1}{2})^2 + \frac{\sqrt{5}-1}{2} \\ =\ & (\frac{\sqrt{5}-1}{2})(\frac{\sqrt{5}-1}{2} + 1) \\ =\ & (\frac{\sqrt{5}-1}{2})(\frac{\sqrt{5}+1}{2}) \\ =\ & \frac{4}{4} = 1 \end{aligned}$$

[6]As in Observation 2.2, we are not attempting to meet deadlines in this schedule.

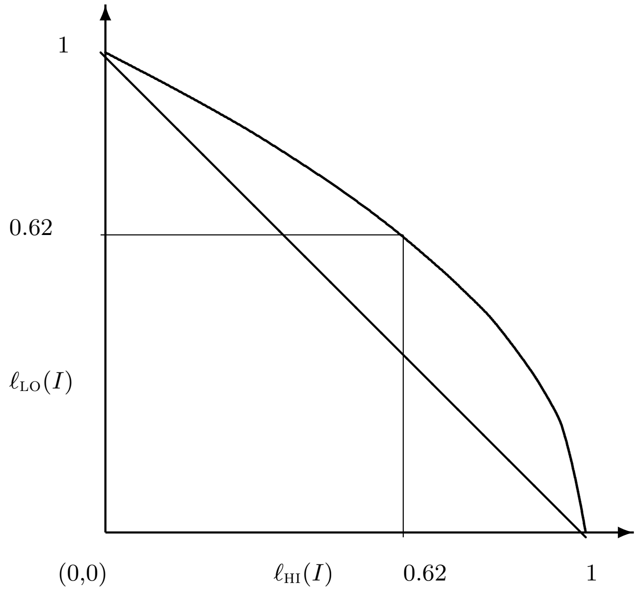

Figure 1: Bound on the LO-criticality load (ℓ_{LO}) as a function of HI-criticality load (ℓ_{HI}).

and such an I hence satisfies Condition 1 of Theorem 2.

5. RELATIONSHIP TO PRIOR APPROACHES

To better explain how the result of Theorem 2 compares to prior work, we have plotted in Figure 1 the bound on the LO-criticality load of an instance I as a function of its HI-criticality load $\ell_{\text{HI}}(I)$, in order for I to be successfully scheduled.

- The straight line connecting the points $(1, 0)$ and $(0, 1)$ has equation $\ell_{\text{LO}}(I) + \ell_{\text{HI}}(I) = 1$, and represents the schedulability condition for the space-time partitioning and other reservations-based approaches: since we must reserve a fraction $\ell_{\text{HI}}(I)$ of the processor for HI-criticality tasks in such an approach, that leaves a fraction $1 - \ell_{\text{HI}}(I)$ of the processor capacity for accommodating additional LO-criticality jobs. Hence any instance that maps on to a point below this line is guaranteed schedulable by space-time partitioning.

- The square with vertices at $(0, 0)$, $(0, 0.62)$, $(0.62, 0.62)$, and $(0.62, 0)$ represents the schedulability condition that is implied by prior work [7]: any instance that maps on to a point within this square is guaranteed, by the results in [7], to be schedulable by OCBP.

- The curve that connects the points $(1, 0)$ and $(0, 1)$ has equation $\ell_{\text{LO}}(I)^2 + \ell_{\text{HI}}(I) = 1$, and represents the OCBP schedulability condition derived in Theorem 2. Any instance that maps on to a point beneath this curve is guaranteed, as a consequence of Theorem 2, to be schedulable by OCBP.

It is immediately evident from the figure that the schedulability region for the condition of Theorem 2 strictly dominates the schedulability regions of both space-time partitioning and of the result in [7], in the sense that those schedulability regions are both contained within the schedulability region for the condition of Theorem 2.

As an example illustrating the implications of this fact, suppose that a system designer were designing an embedded system I in which the HI-criticality workload — those subject to certification — is determined to have a load equal to 25% of the processor capacity (i.e., $\ell_{\text{HI}}(I) = 0.25$). The question to be answered is this: how much LO-criticality functionality that does not need certification may be added to this processor?

- Under space-time partitioning, any instance with LO-criticality load not exceeding $(1-0.25)$, or **75**%, of the processor capacity is guaranteed to be schedulable.
- Since 0.25 is $\leq (\sqrt{5}-1)/2$, we may apply the result from [7]. According to this result, any instance with LO-criticality load not exceeding $(\sqrt{5}-1)/2$, or $\approx$ **62**%, of the processor capacity is guaranteed to be successfully scheduled by OCBP.
- Using the result of Theorem 2, we are able to conclude that any instance with LO-criticality load not exceeding $\sqrt{1-0.25}$, or $\approx$ **86**%, is successfully scheduled by OCBP.

We now prove a stronger result that does not directly follow from Theorem 2, showing that any MC instance that is schedulable using space-time partitioning is guaranteed to be schedulable by OCBP:

THEOREM 3. *Any MC instance I that is schedulable using space-time partitioning is also schedulable using OCBP.*

Proof: Suppose that instance I is schedulable on a preemptive unit-speed uniprocessor under space-time partitioning. This fact implies that there it is possible to construct a schedule for the jobs in I in which each job $J_i \in I$ with $\chi_i =$ LO receives C_i units of execution, and each job $J_i \in I$ with $\chi_i =$ HI receives C'_i units of execution.

By the optimality property of EDF on preemptive uniprocessors [11, 10], it follows that EDF would successfully schedule the behavior of I in which each LO-criticality job J_i executes for C_i units, and each HI-criticality job J_i executes for C'_i units. Let us refer to this EDF schedule as S.

We will now show that the OCBP priority assignment algorithm can construct the priority ordering for the jobs in I, in which jobs are ordered according to deadline with later-deadline jobs receiving lower priority[7]. To see why this is the case, consider the latest-deadline job J_i in I. When the OCBP priority assignment algorithm considers this job as a potential lowest-priority job, there are two cases to consider:

- Suppose that J_i is a HI-criticality job ($\chi_i =$ HI). It follows from our previous argument (above) about the existence of the EDF schedule S that J_i gets C'_i units of execution between its release time and its deadline if all other jobs in I execute at greater priority than J_i, and each such job J_j executes for C'_j time units. Hence, the OCBP priority assignment scheme may assign J_i lowest priority.
- Suppose that J_i is a LO-criticality job ($\chi_i =$). Once again from the existence of the EDF schedule S, it follows that J_i gets C_i units of execution between its release time and its deadline if all other jobs in I execute at greater priority than J_i, and each such job J_j executes for C_j time units if $\chi_j =$ LO and C'_j time units if $\chi_j =$ HI. Since C_j is assumed to be $\leq C'_j$ for all j, it follows that J_i gets C_i units of execution between its release time and its deadline if all other jobs in I execute at greater priority than J_i, and each such job J_j executes for C_j time units. Hence, the OCBP priority assignment scheme may assign J_i lowest priority.

We thus see that the latest-deadline job may be assigned lowest priority according to the OCBP priority assignment scheme. Repeatedly going through the above argument on the instance obtained by removing the latest-deadline job, we see that a total ordering may be obtained on all the jobs in I. Therefore, the OCBP priority assignment scheme is able to successfully schedule any MC instance which can be scheduled using space-time partitioning, and out theorem is proved. ■

It is relatively easy to construct MC instances that are OCBP-schedulable, that space-time partitioning cannot schedule in a correct manner. Thus in addition to having a schedulability region on the $\ell_{\text{LO}} \times \ell_{\text{HI}}$ plane that strictly contains the schedulability region of space-time partitioning (as illustrated in Figure 1), it is also the case that OCBP scheduling strictly dominates space-time partitioning.

[7]That is, although OCBP may find some other correct priority ordering if one exists, it will certainly find the deadline ordering if no other priority ordering is correct. Therefore the OCBP priority cannot declare failure on I if I can be scheduled by space-time partitioning.

6. CONTEXT AND CONCLUSIONS

Due to the rapid increase in the complexity and diversity of functionalities that are performed by safety-critical embedded systems, the cost and complexity of obtaining certification for such systems is fast becoming a serious concern [4]. We believe that in mixed-criticality systems, these certification considerations give rise to fundamental new resource allocation and scheduling challenges which are not adequately addressed by conventional real-time scheduling theory. In prior work [7], we have therefore proposed a job model that is particularly appropriate for representing mixed-criticality workloads, and have studied basic properties of this model. We had also derived an algorithm, called OCBP, for scheduling such mixed-criticality workloads. In this paper, we have conducted a thorough investigation of the schedulability properties of OCBP. We have obtained quantitative bounds on these schedulability properties, derived a load-based sufficient schedulability condition, and demonstrated that OCBP scheduling is superior to the space-time partitioning approach to scheduling in mixed-criticality systems.

7. ACKNOWLEDGEMENTS

This research has been supported in part by AT&T, IBM, and Sun Corps.; NSF grants CNS 0834270 and CNS 0834132; ARO grant W911NF-09-1-0535; and AFOSR grant FA9550-09-1-0549.

8. REFERENCES

[1] ARINC. ARINC 653-1 Avionics application software standard interface, October 2003.

[2] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, England, 1991.

[3] N. C. Audsley. *Flexible Scheduling in Hard-Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1993.

[4] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. S. P. Stanfill, D. Stuart, and R. Urzi. White paper: A research agenda for mixed-criticality systems, April 2009. Available at http://www.cse.wustl.edu/˜ cdgill/CPSWEEK09_MCAR.

[5] S. Baruah. Mixed criticality schedulability analysis is highly intractable. Available at http://www.cs.unc.edu/˜baruah/Pubs.shtml, 2009.

[6] S. Baruah, H. Li, and L. Stougie. Mixed-criticality scheduling: improved resource-augmentation results. In *Proceedings of the ICSA International Conference on Computers and their Applications (CATA)*. IEEE, April 2010.

[7] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. IEEE, April 2010.

[8] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008. IEEE Computer Society Press.

[9] D. de Niz, K. Lakshmanan, and R. R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the Real-Time Systems Symposium*, pages 291–300, Washington, DC, 2009. IEEE Computer Society Press.

[10] M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[11] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[12] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2000.

[13] R. Pellizzoni, P. Meredith, M. Y. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed criticality in SoC-based real-time embedded systems. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, Grenoble, France, 2009. IEEE Computer Society Press.

[14] S. 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.

[15] J. Windsor and K. Hjortnaes. Time and space partitioning in spacecraft avionics. *Space Mission Challenges for Information Technology, IEEE International Conference on*, pages 13–20, 2009.