On Partitioned Scheduling of Fixed-Priority Mixed-Criticality Task Sets [*]

Owen R. Kelly Hakan Aydin Baoxian Zhao
Department of Computer Science
George Mason University
Fairfax, VA 22030 - USA
okelly@gmu.edu aydin@cs.gmu.edu bzhao@gmu.edu

Abstract— **Mixed-criticality real-time systems, where tasks may be associated with different criticality and assurance levels, have attracted much attention in the recent past. In this paper, we consider partitioning-based multiprocessor scheduling of mixed-criticality real-time task sets. Guaranteeing feasibility in this setting is shown to be NP-Hard. With a focus on fixed-priority preemptive scheduling on each processor, we identify the two main aspects of the problem, namely the *task allocation* and *priority assignment* dimensions. For the task allocation dimension, we propose and compare bin-packing-inspired heuristics, based on offline task ordering according to utilization and criticality. For the priority assignment dimension, we compare the well-known Rate Monotonic priority assignment policy with Audsley's priority assignment algorithm. Through simulations, we also assess and discuss the relative importance of these two primary dimensions on the overall mixed-criticality feasibility problem for multiprocessor platforms.**

Keywords- real-time scheduling, multiprocessor systems, mixed-criticality, partitioning-based scheduling, fixed-priority scheduling

I. INTRODUCTION

Hard real-time systems – systems where guaranteeing *temporal predictability* even under worst-case scenarios is of the utmost importance, have received significant attention in research efforts over the past several decades. As a result, there exists a well-established theory for both uniprocessor and multiprocessor hard real-time systems. In particular, the hard real-time scheduling of *periodic* tasks, common in real-time applications, has been the subject of intensive research [7][17].

For periodic real-time applications executed on a single processor, *priority-driven* scheduling has been recognized as an effective framework to guarantee the timing constraints of tasks [17]. In fixed-priority scheduling, each task is assigned a fixed priority that is determined statically. Among fixed-priority algorithms, *Rate Monotonic Scheduling (RMS)*, where the priority of a task is inversely proportional to its period, is known to be optimal for task sets where the relative deadlines are equal to the periods [16]. Among dynamic-priority policies, the *Earliest-Deadline-First (EDF)* policy is known to be optimal [16].

For multiprocessor real-time systems, existing approaches typically fall in the *partitioning* or *global-scheduling* classes [17]. In global scheduling, there is effectively a single queue of ready jobs that are allocated across multiple processors, and hence, a task can migrate from one processor to another at run-time. In contrast, in partitioning-based approaches, each task is statically allocated to a single processor. While global approaches can offer better processor utilization for some task sets, they can also incur greater run-time overhead and cache affinity problems due to potential migrations. In addition, it has been shown that some task sets are not schedulable with a global approach despite having low overall utilization [10]. An important advantage of partitioning-based approaches is that the well-established uniprocessor scheduling theory is applicable to the scheduling of the subset of tasks allocated to individual processors [18].

More recently, there has been a growing interest in analyzing real-time systems that execute tasks of different criticalities. Recent work on these so-called *mixed-criticality* systems has included the development of system models that explicitly take *task criticality* into account [4][5][12][20][21]. Much work in this area stems from the observation that several important classes of real-time applications include different functionalities at varying levels of importance, requiring different *assurance* levels. In particular, these settings mandate that the deadlines of higher-criticality tasks be guaranteed with a higher degree of assurance than those of lower-criticality tasks.

In traditional (i.e., not mixed-criticality) hard real-time models, a single worst-case execution time (WCET) estimate is associated with each task and is used when determining if a task set is schedulable. As noted in [21], this often leads to system under-utilization – the designers have to choose the most conservative WCET estimate for *every* task, regardless of criticality levels. In practice, even for a given task, a different WCET estimate may be available for each criticality level: Larger estimates for higher criticality levels can be obtained through exhaustive analysis, even though the probability of observing these larger WCETs at run-time may be quite low. Conversely, smaller WCET estimates can be obtained for lower criticality levels through less-rigorous methods, for example by considering the average value observed in real system traces.

As a result, a main feature of the mixed-criticality models explored in [21][14][15][4] and adopted in this paper is the specification of a WCET function for each task, i.e., the specification for each task of a WCET corresponding to each criticality level in the system. It is intended that each WCET estimate have a level of assurance that is appropriate for its

[*] This work was supported by U.S. National Science Foundation Awards CNS-1016855 and CNS-0546244 (CAREER award).

corresponding criticality level. Fundamental in the application of this model is the following principle [5][21]: *In assessing the delay that can be experienced by a given task τ, the feasibility analysis must consider, for higher-priority tasks, the WCET values corresponding to the same level of criticality (i.e., assurance) as that of τ.*

This principle has important implications for both real-time systems theory and practice. First, by allowing the use of looser (less pessimistic) WCET estimates when determining the timeliness of low-criticality tasks during feasibility analysis, system utilization can be significantly improved [21]. Second, a comprehensive theory of mixed-criticality system analysis eventually may enable formal *certification* by civilian authorities [4].

However, the development of such a comprehensive new theory poses several challenges. For example, in his seminal paper, Vestal observed that rate-monotonic priority assignment is not optimal for fixed-priority mixed-criticality single-processor scheduling [21]. Consequently, he proposed the use of Audsley's priority assignment algorithm [2], despite the relatively high complexity associated with it. In another negative result, it has been shown that the general problem of mixed-criticality feasibility analysis is NP-Hard in the strong sense on one processor, even for settings where the jobs have the same ready time and there are only two distinct criticality levels [4]. However, the practical promise of the framework is high enough that additional research efforts need to be (and have been, [4][5][12][20][21]) applied to the problem, even though they may be forced, in the final analysis, to offer only "approximate" solutions. More details about related work in the area can be found in Section VI.

This research effort has been partly motivated by the observation that much of the existing and growing literature on mixed-criticality systems targets uniprocessor systems, yet many real-time applications are expected to, and do already run on multi-processor and emerging multi-core platforms. Consequently, we believe a preliminary analysis of mixed-criticality systems for multiprocessor settings will prove useful.

There are two dimensions in which one can explore different approaches and their effect on the overall schedulability of mixed-criticality task sets with fixed priority assignments. The *priority assignment* dimension, to guarantee the feasibility of fixed-priority task sets on uniprocessor systems, is already a difficult problem [21]. While RMS priority assignment is simple and well-known, an optimal solution can be generated by the computationally-expensive priority assignment algorithm developed by Audsley [2]. The consideration of *task allocation* on multi-processor platforms adds a new dimension and simultaneously makes the problem intractable (even for the special case where all tasks have the same criticality). *Bin packing* heuristics such as First-Fit and Best-Fit have been used extensively in the literature for the traditional partitioning problem, especially after first ordering tasks according to *utilization* [17][18]. Alternatively, in mixed-criticality settings, grouping tasks according to criticality (as opposed to utilization) on separate processors appears to be an intuitive solution, though its impact on feasibility is not immediately clear.

The primary focus of this work is the exploration of these two dimensions, i.e., *task allocation* and *priority-assignment strategies* for periodic fixed-priority mixed-criticality task sets. Specifically, we identify the main algorithmic solutions for each dimension and compare them experimentally. Moreover, we undertake an evaluation of the ***relative importance*** of each of these dimensions experimentally. For example, we believe it is interesting to ask whether the use of a criticality-aware task allocation heuristic eliminates or reduces the need for running Audsley's priority assignment algorithm on each processor, or if the use of Audsley's algorithm reduces the need for a criticality-aware allocation heuristic.

The remainder of this paper is organized as follows. In Section II, we present our model and notation. Section III re-visits the fundamental principles of feasibility analysis for fixed-priority mixed-criticality task sets on uniprocessor systems. Section IV elaborates on the two dimensions of our problem (task allocation and priority assignment) and illustrates the trade-offs involved in the multi-processor case through examples. Section V presents our experimental evaluation. Related work and concluding remarks are provided in Sections VI and VII, respectively.

II. System Model and Notation

We consider a task set Γ that is comprised of *n* independent, periodic, mixed-criticality tasks $\tau_1, \ldots, \tau_n$. The period of task τ_i, which is equal to the relative deadline of its jobs, is denoted by T_i. We assume there are $k \leq n$ distinct criticality levels in the system, and that L_i denotes the criticality level of task τ_i (larger L_i values indicate higher criticalities).

Following the convention in existing mixed-criticality research, we assume that the WCET values for task τ_i are specified as a function, C_i , where $C_i(X)$ corresponds to the WCET estimate of τ_i at criticality level X. $C_i(X)$ is assumed to be no smaller than $C_i(X-1)$ [5]. Similarly, the *utilization* values of task τ_i are specified as a function, $u_i(\,)$, where $u_i(X)$ denotes the utilization of τ_i at criticality level X and is defined as the ratio of $C_i(X)$ to T_i. We define the *nominal utilization* of task τ_i to be the value of its utilization function at its criticality level, i.e., $u_i(L_i)$.

Tasks are scheduled using a partitioning-based approach on a set of *m* homogeneous processors identified as $\{P_1, \ldots, P_m\}$. The subset of tasks allocated to processor P_i is denoted by Γ_i. On each processor, tasks are scheduled by a preemptive fixed-priority scheduling algorithm. Task priorities are identified by positive integers beginning with 1, which represents the highest priority. The priority of task τ_i is denoted by ρ_i.

III. Preliminaries

In this section, we overview the basics of fixed-priority mixed-criticality feasibility analysis on uniprocessor systems, since it forms the basis of our partitioned-scheduling framework after the task allocation phase. Specifically, we use the mixed-criticality feasibility analysis methodology introduced in [21]. In particular, when evaluating whether a periodic task will meet its deadlines in all its instances, the

WCET values used in the representation of higher-priority tasks must be of the same criticality level as that of the task under evaluation [5][21]. We determine whether a set of tasks allocated to a processor is schedulable given a particular fixed priority assignment by checking whether the worst-case response time for each task is less than or equal to its relative deadline. The worst-case response time of a task is incurred by a job that is released at a *critical instance* of the task, that is, by a job of the task that is released at the same time as jobs of all higher-priority tasks [16]. As described in [21], the worst-case response time R_i of task τ_i is therefore the smallest fixed-point solution of:

$$R_i = \sum_{j:\rho_j \le \rho_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i) \tag{1}$$

Notice that, in the computation of the worst-case response time of task τ_i, the interference of higher-priority tasks is evaluated at the WCET values that correspond to the criticality level L_i [21]. Therefore, the *assignment of priorities* to individual tasks is of utmost importance in mixed-criticality scheduling [21].

Example 1: Consider two tasks, τ_1 and τ_2, running on a uniprocessor system with the following parameters:

τ_1: T_1=20, $C_1(1)$=4, $C_1(2)$=16, L_1=1

τ_2: T_2=50, $C_2(1)$=12.5, $C_2(2)$=17.5, L_2=2

With RM priority assignment, task τ_1 has the higher priority despite having the lower criticality level – a situation referred to as *criticality inversion* in [20]. Because τ_2 has high criticality, its worst-case response time is calculated using the high-criticality WCET of each task. The task diagram in Fig. 1 represents the evaluation of the schedulability of τ_2 with RM priority assignment under critical-instant phasing. With $C_1(2)$ used as the WCET for τ_1, the response time of τ_2 exceeds its deadline. As a result, the task set is not schedulable with RM priority assignment.

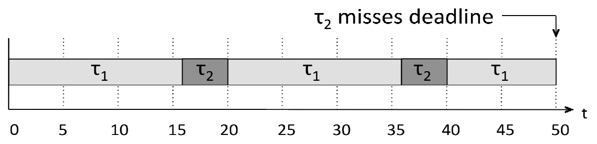

Figure 1. RM Priority Assignment Results in Criticality Inversion

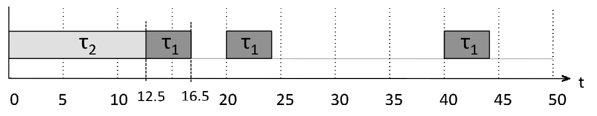

Figure 2. Alternative Priority Assignment

Consider next the case in which τ_2 is assigned higher priority than τ_1 (Fig. 2). Because τ_1 has low criticality, $C_2(1)$ is used when determining the response time of τ_1. As shown in Fig. 2, the task set is schedulable with this priority assignment. Note that, if the feasibility of this task set were considered using a traditional model, with the larger, more-pessimistic WCET estimate used for each task, then the task set would not be schedulable under either of the two possible priority assignments. This simple example illustrates how the use of a mixed-criticality model, combined with proper priority assignment, can offer increased processor utilization.

IV. Dimensions of Mixed-Criticality Fixed-Prioity Partitioned Scheduling

Our focus in this paper is the problem of scheduling a mixed-criticality task set on a fixed number of processors such that the tasks are statically allocated to processors, and such that the subset allocated to each processor is scheduled by a preemptive, fixed-priority algorithm in a feasible manner. In this section, we identify the two main dimensions of the problem and the algorithmic solutions applicable in each. Then we characterize the integrated schemes that can be formed through a combination of the solutions proposed for each dimension. We first formally define the *Mixed-Criticality Fixed-Priority Partitioning* problem (MCFP–PARTITION).

Problem MCFP-PARTITION: Partition a given set of periodic, independent, mixed-criticality tasks, $\tau_1 \ldots \tau_n$, with relative deadlines equal to periods, upon *m* homogeneous multiprocessors, and determine fixed priorities for each, such that the set of tasks allocated to each CPU is feasible.

Proposition 1: *MCFP–PARTITION is NP-Hard in the strong sense.*

Proposition 1 follows from the classical result on the intractability of scheduling a set of *n* real-time tasks on *m* processors even when the tasks have identical criticality levels and share the same release time and deadline [11].

This result implies that one has to work with heuristic solutions to tackle this problem, unless $P = NP$. To better explore the possibilities, we consider the two main dimensions of the problem separately:

1. **Task Allocation.** Partition the tasks $\tau_1 \ldots \tau_n$ on the processor set $P_1 \ldots P_m$.
2. **Priority Assignment.** For i ∈ {1, …, m}, given the subset of tasks Γ_i allocated to P_i, assign a priority level ρ_j to each task $\tau_j \in \Gamma_i$.

A. Task Allocation

The bin-packing-inspired partitioning heuristics that have been applied to the multiprocessor scheduling of traditional task sets [17][18] can also be applied in the mixed-criticality context. These include:

First-Fit (FF) – The order of processors is fixed and each task is allocated to the first processor on which it "fits", i.e., on which it can be successfully scheduled along with the other tasks already allocated to that processor.

Best-Fit (BF) – Each task is allocated to the processor with the smallest unused capacity among those processors on which it fits.

Worst-Fit (WF) – Each task is allocated to the processor with the largest unused capacity among those processors on which it fits.

When these heuristics are applied to traditional task sets, the determination of whether a subset of tasks is schedulable on a processor is a matter of applying the schedulability analysis results derived for traditional task sets. When applied to mixed-criticality task sets, schedulability is determined in the context of the chosen mixed-criticality system model – in our case by considering the response-time formula (1).

In bin-packing-inspired partitioning techniques, such as FF, BF, and WF, it has been shown that ordering tasks first according to a certain criterion, and then processing them in that order, generally improves performance. For example, ordering tasks according to *decreasing utilization* values proves helpful [17][18]. The resulting heuristics are referred to as First-Fit-Decreasing (FFD), Best-Fit-Decreasing (BFD), and Worst-Fit-Decreasing (WFD), respectively. We consider two options for ordering tasks prior to the application of the traditional partitioning heuristics:

Decreasing Utilization (DU): In this approach, "large" tasks with high utilization values are allocated first, in a way that is similar to the application of algorithms such as FFD, BFD, and WFD to traditional task sets. However, because each mixed-criticality task is associated with multiple utilization values (one for each criticality level), such an ordering requires that a single utilization value be identified for each task. In this paper, when ordering mixed-criticality tasks by utilization, we use each task's *nominal utilization*, i.e., $u_i(L_i)$ – the value of the task's utilization function at the criticality level of the task.

Decreasing Criticality (DC): Tasks are ordered according to criticality, and tasks at the same criticality level are further ordered by decreasing nominal utilization. This approach has the advantage of grouping tasks with identical or similar criticalities on the same processor. Hence, this approach reduces instances of *criticality inversion* (described in [20]), which occur when a low-criticality task is assigned higher priority than a high-criticality task. However, allowing the assignment of tasks of different criticalities to the same processor may, in some cases, offer improved processor utilization.

B. Priority Assignment

The objective in this dimension is the assignment of a priority to each task such that, for each processor, the subset of tasks allocated to the processor is schedulable given the assigned priorities. That is, the priorities must be chosen such that the response time computed for each task through (1), on its assigned processor, is no larger than the task's deadline. In this paper, we consider two priority-assignment schemes:

RM Priority Assignment **([16])** – In this solution, higher priorities are assigned to tasks with smaller periods. Because it is optimal among fixed-priority algorithms for traditional task sets in which relative deadlines are equal to periods, RM scheduling is widely used. Despite the fact that it is not optimal for mixed-criticality task sets (noted in [21] for the more-general case of Deadline-Monotonic scheduling), its simplicity warrants its consideration. The complexity of RM assignment is $O(n_i \log n_i)$ on each CPU, where n_i is the number of tasks allocated to processor P_i, since one needs to order tasks according to their periods before the priority assignment.

Audsley's Optimal Priority Assignment algorithm **([2])** – Priorities are assigned in order, from lowest to highest. In each iteration, the algorithm determines if some task can be assigned the next priority. If so, it assigns that priority, removes the task from the subset of tasks that have not yet received an assignment, then recursively assigns the remaining priorities to the remaining subset of tasks in the same manner. If, in some step, an appropriate task cannot be found for the next priority level, the task set is not schedulable with a fixed-priority algorithm. Audsley's algorithm has the advantage of being optimal for mixed-criticality task sets as well as for traditional task sets [21]. A disadvantage, however, is its increased complexity relative to RM scheduling – The algorithm requires $O(n_i^2 + n_i)$ tests (where n_i is the number of tasks allocated to P_i), and in each test feasibility is checked using Time-Demand Analysis, which may take pseudo-polynomial time [17].

C. Integrated Schemes

Given the two main options in each of the two primary dimensions of the problem, for a given partitioning heuristic such as FF, one can obtain a total of four schemes to address the mixed-criticality partitioning problem, which we denote by ***DU-RM, DU-Audsley, DC-RM*** and ***DC-Audsley***. Here, the first component of the scheme (*DU* or *DC*) indicates whether the tasks are ordered according to utilization or criticality prior to task allocation. The second component (*RM* or *Audsley*) refers to the priority assignment strategy on each CPU. Obviously, each of these options has its own complexity and performance figures. For example, Audsley's priority assignment scheme is optimal on a given CPU, but its complexity is much higher than the simple RM assignment. Also, it is not immediately clear whether ordering tasks according to utilization or criticality is always the better choice. Next, we provide two examples to illustrate how the potential choices may impact feasibility.

Example 2: Consider the task set given in Table I, to be scheduled on two processors, P_1 and P_2. Assume that, for task allocation, we use the First-Fit (FF) heuristic.

TABLE I. EXAMPLE TASK SET

	T_i	$C_i(1)$	$C_i(2)$	$u_i(1)$	$u_i(2)$	L_i
τ_1	20	4	16	**0.2**	0.8	1
τ_2	50	12.5	17.5	0.25	**0.35**	2
τ_3	40	12	18	0.3	**0.45**	2
τ_4	10	4	5	**0.4**	0.5	1

First, consider the DU-RM scheme, where tasks are ordered according to their nominal utilizations (typed in bold in Table I) and RM priorities are used. With DU ordering, the tasks are allocated in the order of τ_3, τ_4, τ_2, and τ_1. Fig. 3 depicts the result of the application of DU-RM to this task set. The critical instant phasing that arises from using RM priorities on each CPU is shown. τ_3 and τ_4 are successfully allocated to P_1. Note that the high-criticality WCET values for both tasks are used when determining whether τ_3 meets its deadlines. Neither τ_1 nor τ_2 fit on P_1, so they need to both be placed on P_2. However, as analyzed in Example 1 (Section III), τ_2 will miss its deadline when it is assigned to the same processor as τ_1 with RM priorities. Therefore, the task set in this example is not schedulable with DU-RM.

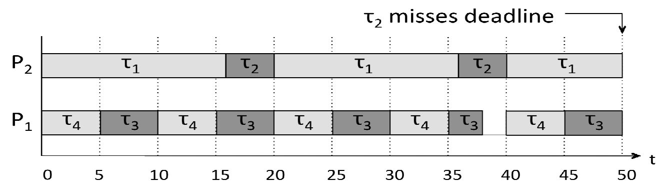

Figure 3. Task Set not Schedulable with DU-RM

However, if τ_2 is assigned higher priority than τ_1 (Fig. 4), then we obtain the second case discussed in Example 1 (Section III), leading to a feasible schedule. This observation effectively implies that the DU-Audsley scheme (by considering all possible priority assignments) would be able to successfully partition the given mixed-criticality task set.

Now consider the addition of a new task, τ_5 , with the following parameters: $T_5 = 15$, $C_5(1) = 3$, $C_5(2) = 4.5$, $L_5 = 2$. With these parameters, τ_5's utilization function has the values $u_5(1) = 0.2$ and $u_5(2) = 0.3$. Fig. 5 shows the result of the application of DU when *any* fixed-priority assignment can be used, i.e., when we are not constrained to use RM priority assignments. This scenario is effectively equivalent to the application of the DU-Audsley scheme on the augmented task set. Considered in the order of decreasing utilization, τ_3 and τ_4 are allocated to P_1; then τ_2 and τ_5 are allocated to P_2. We then consider whether τ_1 can be allocated, with τ_3 and τ_4, to P_1. If τ_1 were assigned lowest priority on P_1, τ_3 would use 12 units of time during τ_1's first period and τ_4 would use 8 units; τ_1 would therefore receive no CPU time. We also find that neither τ_3 nor τ_4 can be assigned lowest priority. Having determined that τ_1 cannot be allocated to P_1, we consider next whether it can be allocated, with τ_2 and τ_5, to P_2. None of these three tasks meets its deadlines if assigned lowest priority. For example, as shown in Fig. 5, τ_1 misses its deadline if assigned lowest priority. We have therefore determined that the task set is not schedulable with DU under any fixed-priority assignment, which implies that DU-Audsley would fail to obtain a feasible partitioning.

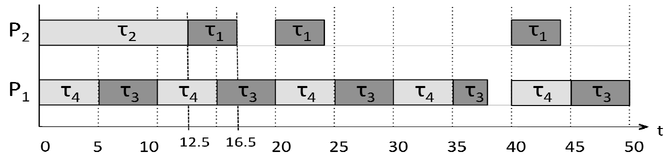

Figure 4. Task Set becomes Schedulable with Change in Priorities on P_2

Now consider ordering these five tasks according to *decreasing criticality* (with ties broken according to decreasing nominal utilization), giving the processing order of τ_3, τ_2, τ_5, τ_4, and τ_1 for the First-Fit technique. Further, assume the simple priority assignment RM on each CPU, yielding the integrated scheme DC-RM. We consider whether the example task set (augmented with τ_5) is schedulable with DC-RM. The result of the application of DC-RM is shown in Fig. 6. With RM priority assignments on each CPU, the task set is schedulable.

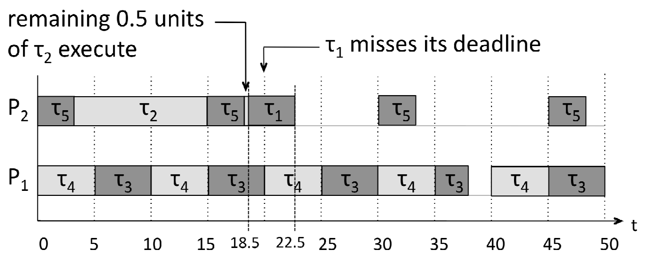

Figure 5. Addition of Task τ_5

Example 2 demonstrates that sometimes, by simply modifying the task ordering scheme (in this case by switching from DU with Audsley's optimal priority assignment to DC with simple RM priority assignment), one can successfully partition mixed-criticality task sets. This last result may raise the question of whether ordering tasks according to criticality before partitioning is always preferable to ordering according to task utilization (i.e. the question of whether DC dominates DU). However, the answer is negative, as shown in the next example.

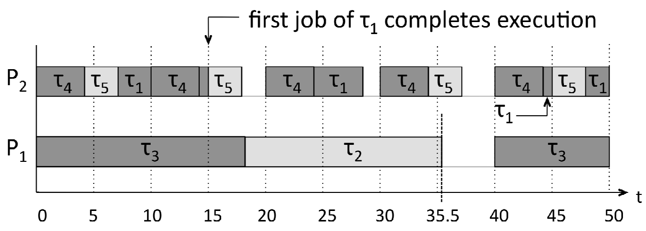

Figure 6. Task Set Schedulable with DC-RM

Example 3: Consider the task set given in Table II. Again, assume the tasks are partitioned using the First-Fit (FF) scheme.

TABLE II. TASK SET SCHEDULABLE BY DU-RM BUT NOT DC-AUDSLEY

	T_i	$C_i(1)$	$C_i(2)$	$u_i(1)$	$u_i(2)$	L_i
τ_1	40	14	24	.35	**.6**	2
τ_2	80	24	32	.3	**.4**	2
τ_3	120	66	72	**.55**	.6	1
τ_4	400	180	200	**.45**	.5	1

The tasks are listed in the order in which they would be allocated by the Decreasing-Criticality (DC) heuristic. We first consider whether they are schedulable by DC-Audsley. In other words, while processing tasks according to decreasing criticality, we will try all possible priority assignments on each CPU in an attempt to find at least one feasible solution. As shown in Fig. 7, tasks τ_1 and τ_2 are successfully allocated to P_1, with τ_1 assigned the higher priority. τ_3 cannot be allocated, with τ_1 and τ_2, to P_1. One way to see this is to note that the sum of the low-criticality utilizations of tasks τ_1, τ_2, and τ_3 is greater than one. τ_3 is therefore allocated to P_2, and τ_4 is considered next. Because the sum of the low-criticality utilizations of tasks τ_1, τ_2, and τ_4 is greater than one, it is clear that τ_4 cannot be allocated to P_1. We also find that, regardless of its priority assignment relative to τ_3, τ_4 cannot be assigned to P_2. The case in which τ_4 is assigned the lower priority is shown in Fig. 7. We therefore conclude that the task set is not schedulable by DC-Audsley.

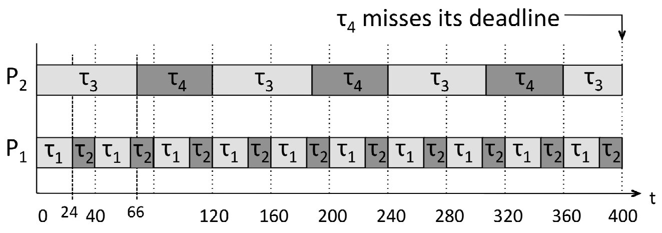

Figure 7. Task Set is not Schedulable by DC-Audsley

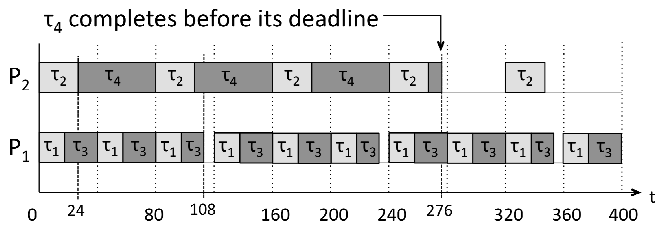

Figure 8. Task Set is Schedulable by DU-RM

Now consider whether the same task set is schedulable if the tasks are processed according to their utilization values, i.e., using the DU heuristic. Furthermore, assume the simple RM priority policy is adopted on each CPU – in other words, assume we do not try all possible priority assignments. The order in which these tasks are considered for allocation by DU is τ_1, τ_3, τ_4, and τ_2. As shown in Fig. 8, tasks τ_1 and τ_3 are successfully allocated to P_1 with RM priority assignments. τ_4 cannot be allocated to P_1 with RM priority assignments and is therefore allocated to P_2. τ_2 cannot be allocated to P_1, but as shown in Fig. 8, it is successfully allocated to P_2 when RM priority assignments are employed. The task set is therefore schedulable by DU-RM, despite not being schedulable by DC-Audsley – the more sophisticated Decreasing Criticality ordering heuristic that tries all possible priority assignments on each CPU. Given these examples, we can offer the following important remark:

Remark 1: *The decreasing-utilization (DU) and decreasing-criticality (DC) partitioning heuristics are* incommensurable*, in the sense that there exist task sets that are successfully scheduled by one but not the other.*

V. Experimental Evaluation

In this section, we undertake an experimental evaluation of the four mixed-criticality partitioning heuristics, namely, *DU-RM*, *DU-Audsley*, *DC-RM*, and *DC-Audsley.* In addition to establishing the relative ordering of these schemes over a broad spectrum of system parameters, we are interested in *assessing the impact of choices in each dimension (i.e., task allocation vs. priority assignment)* on overall scheduling performance.

For that purpose, a simulator was developed in the Java programming language. For a system with a maximum of *k* criticality levels, the criticality level of each task was determined randomly between 1 and *k*. In simulation studies for traditional, single-criticality systems, the *load* index is often controlled by generating a task set for a target total utilization. For mixed-criticality systems, as each task has *k* utilization levels corresponding to *k* criticality levels, we chose to control the load of the system by varying the sum, denoted by U_{tot}^{max}, of each task's utilization at the highest criticality level. In other words, $U_{tot}^{max} = \Sigma\, u_i(k)$. For a given U_{tot}^{max} value, we generated the $u_i(k)$ values (utilizations at the maximum criticality level) for a task set using the UUnifast-Discard algorithm proposed in [8]. For a given task, once $u_i(k)$ was identified, the utilization values at intermediate criticality levels were generated using a uniform distribution between $0.4*u_i(k)$ and $u_i(k)$.

The minimum and maximum task periods were set to 1 and 1000 ms, respectively. As in [8], task periods were generated using a log-uniform distribution. Results were generated for 1, 2, 4, and 8 criticality levels, and for 4 and 8 CPUs. For each U_{tot}^{max} value, 1000 task sets were generated and the scheduling of each task set was attempted with each of the four mixed-criticality partitioning heuristics. As we considered 4- and 8-CPU systems separately, we present the results as a function of the *normalized* U_{tot}^{max}, which is defined as the ratio of U_{tot}^{max} to the number of CPUs – in other words, it is given by the quantity (U_{tot}^{max}/m). We considered task sets containing 40 and 60 tasks, separately.

In Fig. 9, we present the percentage of task sets successfully scheduled (success ratio) for task sets with 40 tasks and 4 criticality levels, scheduled upon 4 CPUs, as a function of the load (normalized U_{tot}^{max}). We first observe that the percentage of task sets successfully scheduled decreases with increasing load for all the schemes. This is in accordance with the existing theory [18], which indicates that with increasing load the likelihood of finding a feasible partition decreases, even for single-criticality systems. However, it is interesting to note that even at normalized U_{tot}^{max} values equal to or slightly exceeding 1 (i.e., when $U_{tot}^{max} \approx m$), almost all the task sets are scheduled successfully. This is due to the flexibility provided by the mixed-criticality feasibility test: Even though a task's utilization at the maximum criticality level (that contributes to U_{tot}^{max}) may be large, the specific utilization value that is considered in the feasibility test (in other words, its *effective utilization*) depends on its own criticality and the criticality levels of tasks that are assigned *lower* priority and allocated to the same CPU. Consequently, we observe that in particular, with 4 criticality levels and optimal priority assignment (which can be obtained through Audsley's algorithm), the performance is almost 100% even for normalized U_{tot}^{max} values around 1.2. With increasing load, the success ratio drops, but still remains at or above 50% for the DC-Audsley and DU-Audsley schemes when the normalized U_{tot}^{max} does not exceed 1.7.

As to the relative performance of the schemes, we observe that for a given task-ordering policy (DC or DU), employed before the partitioning phase, using Audsley's priority assignment algorithm on each CPU offers significantly improved performance relative to using RM priority assignment. This is not surprising, given that Audsley's algorithm is optimal in this setting, while RMS is not. Moreover, the very fact that DU-Audsley outperforms DC-RM throughout the entire spectrum suggests that the *selection of the priority assignment scheme is more critical than the selection of the task ordering algorithm used prior to partitioning.*

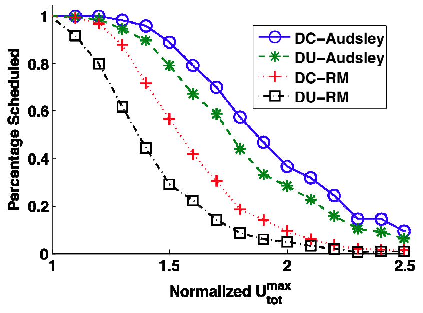

Figure 9. Impact of Normalized U_{tot}^{max} (4 CPUs, 40 Tasks)

However, for a given priority assignment scheme (RM or Audsley), we observe that ordering tasks according to criticality gives better performance than ordering according to utilization in this setting. Even though the offline ordering of tasks according to utilization is known to improve schedulability for the general real-time partitioning problem [18], these results suggest that avoiding or reducing *criticality inversions* by grouping tasks according to criticality proves more important in the average case. With DU ordering, low-criticality tasks are more likely to be co-allocated with high-criticality tasks on the same processor, and depending on the specific period values, a large number of criticality inversions may be unavoidable.

Fig. 10 presents the results for the same setting (4 CPUs and 4 criticality levels), but with 60-task sets instead of 40-task sets. The relative ordering of the schemes remains the same, but one notable difference is that the performance of the DU schemes slightly worsens compared to the trends seen in Fig. 9 with 40-task sets. This is primarily due to two effects: First, with a larger number of tasks allocated to the same number of CPUs, the average number of tasks per CPU increases and the negative impact of each criticality inversion (which occurs more commonly with the DU schemes) is more pronounced. Second, the average size (utilization) of each task decreases, and the potential benefits of ordering tasks according to utilization become less important for feasibility [18].

The plots in Fig. 11 and Fig 12. were generated with the same parameter settings used for Fig. 9 and Fig. 10, respectively, except that the number of CPUs was increased from 4 to 8. Because the number of tasks per set was held constant (at 40 and 60), this increase in the number of CPUs results in a decrease in the average number of tasks per CPU. Consequently, criticality inversions have less impact in terms of an increase in effective utilization. In fact, looking at Fig. 11, we notice that the DU schemes now perform as good as, and in fact, *slightly better* than the corresponding DC schemes when the number of tasks per CPU is smallest. With a slight increase in the number of tasks per CPU, as seen in Fig. 12, the DC schemes start to outperform the DU schemes (for a given load and priority assignment policy). Consequently, we can state that *with an increase in the number of tasks per CPU, a reduction in the number of criticality inversions through a decreasing-criticality (DC) ordering becomes more important for the improvement of schedulability.* In fact, when this ratio is lower than a certain threshold, the DU schemes perform as good as the corresponding DC schemes.

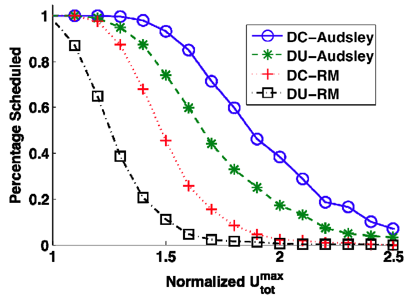

Figure 10. Impact of Normalized U_{tot}^{max} (4 CPUs, 60 Tasks)

In our analysis so far, we have assumed 4 criticality levels. Next, we analyze the impact of varying the number of criticality levels. In Fig. 13, results are plotted for the four schemes as a function of the number of criticality levels, with task sets with 40 tasks each scheduled on 4 CPUs, and with a normalized U_{tot}^{max} value set to 2 (U_{tot}^{max} = 2*4 = 8) . Recall that individual task utilizations, for a given U_{tot}^{max} value, are generated by considering the number of distinct criticality levels, *k*. Task utilizations at the maximum criticality level are chosen such that their sum is equal to U_{tot}^{max} – in this case equal to 8, and task utilization values at intermediate criticality levels are generated randomly, but such that they decrease with decreasing criticality level.

As a result, when there is only one criticality level, the generated task sets correspond to traditional (single-criticality) task sets with total utilization twice the number of CPUs (U_{tot}^{max} = 8). Consequently, none of the generated task sets can be partitioned in a feasible manner at this high load value (as suggested in [18]) when there is only one criticality level. But when the number of criticality levels is increased to 2, the criticality levels of some tasks drop from the maximum level (2) to 1, and this allows for the possibility that their effective utilizations drop in feasibility tests. This results in non-zero success ratios for all the schemes (and highlights the motivation for mixed-criticality levels). With 4 criticality levels, DC-Audsley is able to schedule 35% of the task sets, followed next in performance by DU-Audsley. At this high normalized U_{tot}^{max} level (of 2) the schemes using the RM priority assignment can schedule only 5-10% of the task sets.

Results are plotted in Fig. 14 for the same parameters as Fig. 13, except that task sets are scheduled on 8 CPUs rather than 4. Normalized U_{tot}^{max} is still 2, implying that U_{tot}^{max} = 2 * 8 = 16. As discussed previously, this increase in the number of CPUs, while other parameters (including normalized U_{tot}^{max}) are kept constant, results in a decrease in the number of tasks per CPU as well as an increase in the average task utilization. Consequently, the number of criticality inversions decreases, the performance of all schemes improves, and moreover, the performance of the DU schemes becomes comparable to that of the DC schemes.

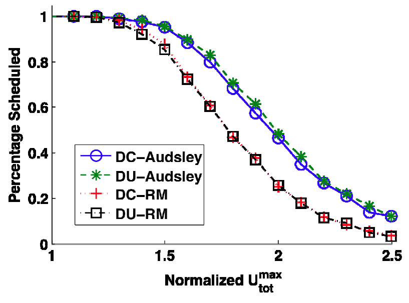

Figure 11. Impact of Normalized U_{tot}^{max} (8 CPUs, 40 Tasks)

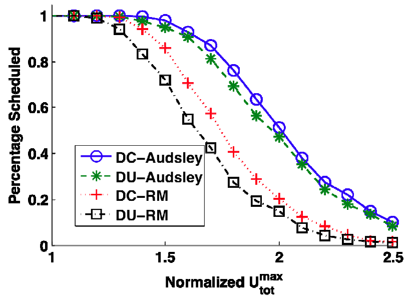

Figure 12. Impact of Normalized U_{tot}^{max} (8 CPUs, 60 Tasks)

The heuristics evaluated in these results allocate tasks according to the First-Fit (FF) algorithm after ordering tasks according to criticality or utilization. We also repeated experiments to compare this choice (FF) against other well-known heuristics, namely Best-Fit (BF) and Worst-Fit (WF). Specifically, we implemented versions of DU-Audsley, DC-Audsley, DU-RM, and DC-RM that use BF and WF, as well as FF. Providing all the plots for the resulting 12 combinations is unrealistic given space constraints. We therefore present, for BF and WF, only the results for DC-Audsley and note that the results are fairly consistent for the other three schemes. Fig. 15 compares the performance of DC-Audsley when implemented with the FF, BF, and WF allocation heuristics, for the case where the parameters are the same as those used for Fig. 9. As shown, the results from the FF and BF heuristics are almost the same, and the results from WF are worse than those obtained with FF or BF. These relative performance results for FF, BF, and WF are consistent with those obtained with traditional system models [18]. In Fig. 16, results are shown for the case in which parameters are the same as those used for Fig. 15, except that the number of CPUs is increased from 4 to 8. As shown, the relative performance of the three heuristics remains the same as the number of CPUs is increased. These results, compounded with the fact that FF is simpler to implement than BF, suggest that it is reasonable to select First-Fit as the basis for the evaluated partitioning heuristics.

VI. Related Work

In [21], Vestal proposed a model for mixed-criticality systems that explicitly takes criticality into account. In the same paper, the use of Audsley's Optimal Priority Assignment algorithm [2] is suggested and is applied using the proposed mixed-criticality model as the basis for schedulability analysis. Period transformation is also considered in [21]. In [5], using the model proposed in [21], it is shown that EDF is not optimal for the scheduling of mixed-criticality task sets. There are mixed-criticality task sets that are not schedulable by EDF, but that are schedulable if job-level, dynamic priorities are used. Also, EDF and RM are incomparable – there are mixed-criticality task sets schedulable by EDF but not RM, and there are others that are schedulable by RM but not by EDF. A hybrid scheduling approach is proposed in [5] that dominates both EDF and RM. The model proposed in [21] is explored further in [3] and is considered in the context of certification of mixed-criticality systems in [4], [14], and [15].

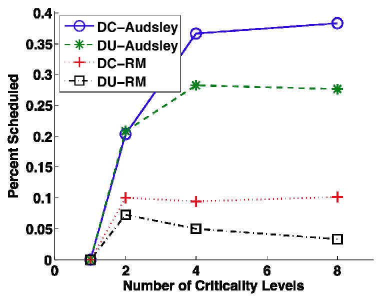

Figure 13. Impact of the Number of Criticality Levels (4 CPUs)

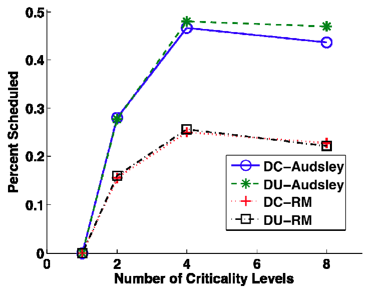

Figure 14. Impact of the Number of Criticality Levels (8 CPUs)

Another approach to the modeling of mixed-criticality systems and the uniprocessor scheduling of such systems is offered in [20], in which each task is assigned a nominal WCET and an overload WCET. A key tenet of the model is the guarantee that a task can execute for the duration of its overload budget (i.e., overload WCET) provided that no higher-criticality task exceeds its nominal WCET. The model proposed in [20] is explored further in [12]. The scheduling of mixed-criticality task sets on multiple CPUs is considered in [13] using the uniprocessor scheduling algorithm proposed in [20], which requires the offline calculation of "zero-slack" instants, and an online component that changes a task's mode at its zero-slack instant. In contrast, we apply traditional, fixed-priority uniprocessor scheduling, but with schedulability analysis performed using the model proposed by Vestal in [21], and our focus is the assessment of variation in partitioning and priority assignment approaches in terms of their impact on schedulability in a mixed-criticality setting. Mixed-criticality, multiprocessor scheduling is also considered in [19], which extends the work presented in [1]. These papers propose a two-level, hierarchical scheduling framework that makes use of container tasks, with a different type of container task applied to the scheduling of tasks at each criticality level. Our focus differs in that we explore variation in task allocation and fixed-priority assignment in the mixed-criticality setting.

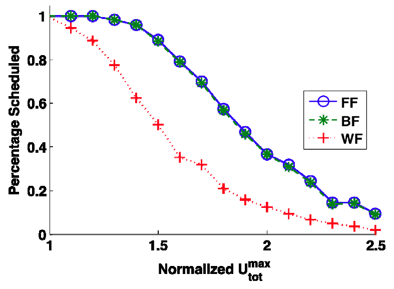

Figure 15. Comparison of Partitioning Heuristics (4 CPUs)

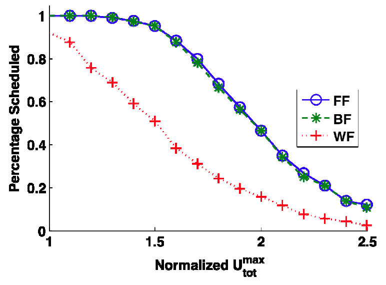

Figure 16. Comparison of Partitioning Heuristics (8 CPUs)

VII. Conclusions

In this paper, we investigated fixed-priority, partitioning-based approaches to the multiprocessor scheduling of mixed-criticality task sets, and assessed the relative importance of the two primary dimensions of the problem - task allocation and priority assignment. We have identified four multiprocessor scheduling algorithms that result from choices made in these two dimensions. For task-allocation, tasks are ordered according to utilization (DU) or criticality (DC) before application of a partitioning heuristic such as First-Fit. For priority assignment on each processor, we considered the simple RM priority assignment, as well as the optimal priority assignment that can be obtained through Audsley's algorithm [2]. We established that the DC and DU task ordering heuristics are incommensurable in the sense that there are tasks sets that can be scheduled by one, but not the other. Our experimental results suggest that in general, using Audsley's priority assignment algorithm offers a significant advantage over RM assignment. We observed that typically DC performs better than DU, but that the difference decreases with a decrease in the number of tasks per CPU.

References

[1] J.H. Anderson, S.K. Baruah, B.B. Brandenburg. "Multicore operating-system support for mixed criticality," Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, April 2009.

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

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, L. Stougie. "Scheduling Real-Time Mixed-Criticality Jobs," Proceedings of the 35th International Symposium, Mathematical Foundations of Computer Science, pp. 90-101, 2010.

[4] S. Baruah, H. Li, L. Stougie. "Towards the design of certifiable mixed-criticality systems," Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 13-22, April 2010.

[5] S. Baruah, S. Vestal. "Schedulability analysis of sporadic tasks with multiple criticality specifications," Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp. 147-155, 2008.

[6] E. Bini, G. Buttazzo. "Measuring the performance of schedulability tests," Real-Time Systems, vol. 30, pp. 129-154, May 2005.

[7] G.C. Buttazzo. Hard Real-Time Computing Systems, 2nd ed. New York, NY: Springer, 2005.

[8] R.I. Davis, A. Burns. "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," Proceedings of the 30th IEEE Real-Time Systems Symposium, pp. 398-409, December 2009.

[9] R.I. Davis, A. Burns. "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems," Technical Report YCS-2009-443, Department of Computer Science, University of York, 2009.

[10] S.K. Dhall, C.L. Liu. "On a real-time scheduling problem," Operations Research, vol. 26, no. 1, pp. 127-140, February 1978.

[11] M.R. Garey, D.S. Johnson. Computers and Intractability, A Guide to the Theory of NP-Completeness. New York, NY: W.H. Freeman and Company, 1979.

[12] K. Lakshmanan, D. de Niz, R. Rajkumar. "Mixed-criticality task synchronization in zero-slack scheduling," Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 47-56, April 2011.

[13] K. Lakshmanan, D. de Niz, R. Rajkumar, G. Moreno. "Resource allocation in distributed mixed-criticality cyber-physical systems," Proceedings of the 30th IEEE International Conference on Distributed Computing Systems, pp. 169-178, June 2010.

[14] H. Li, S. Baruah. "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," Proceedings of the 31st IEEE Real-Time Systems Symposium, pp. 183-192, December 2010.

[15] H. Li, S. Baruah. "Load-based schedulability analysis of certifiable mixed-criticality systems," Proceedings of the 10th ACM International Conference on Embedded Software, pp. 99-107, 2010.

[16] C.L. Liu, J.W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of the ACM, vol. 20, no. 1, pp. 46-61, January 1973.

[17] J.W.S Liu. Real-Time Systems. Upper Saddle River, NJ: Prentice Hall, 2000.

[18] J. M. Lopez, M. Garcia, J. L. Diaz, D. F. Garcia. "Utilization bounds for Multiprocessor Rate-Monotonic Systems", Real-Time Systems, vol. 24, pp. 5 – 28, January 2003.

[19] M.S. Mollison, J.P. Erickson, J.H. Anderson, S.K. Baruah, J.A. Scoredos. "Mixed-Criticality Real-Time Scheduling for Multicore Systems," Proceedings of the 10th IEEE International Conference on Computer and Information Technology, pp. 1864-1871, 2010.

[20] D. de Niz, K. Lakshmanan, R. Rajkumar. "On the scheduling of Mixed-Criticality Real-Time Task Sets," Proceedings of the 30th IEEE Real-Time Systems Symposium, pp. 291-300, December 2009.

[21] S. Vestal. "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," Proceedings of the 28th IEEE International Real-Time Systems Symposium, pp. 239-243, 2007.