ATLANTIS PRESS International Journal of Computational Intelligence Systems, Vol. 11 (2018) 219–237

Criticality-cognizant Clustering-based Task Scheduling on Multicore Processors in the Avionics Domain

K. Nagalakshmi [1], N. Gomathi [2]

[1] *Department of Information Technology, E.G.S. Pillay Engineering College, Nagapattinam, Tamilnadu, India*
E-mail: nagulaxmi@gmail.com

[2] *Department of Computer Science and Engineering, Vel Tech Dr.RR & Dr.SR University, Chennai, Tamilnadu, India*
E-mail: gomathin@veltechuniv.edu.in

Received 15 August 2017

Accepted 26 October 2017

Abstract

Scheduling of mixed-criticality systems (MCS) on a common computational platform is challenging because conventional scheduling approaches may cause inefficient utilization of shared computing resources. In this paper, we propose an approach called Clustering-based Partitioned Earliest Deadline First (C-PEDF) algorithm to schedule dual-criticality implicit-deadline sporadic tasks on a homogeneous multicore system. Our C-PEDF scheduling approach exploits (i) a Clustering-based bin-packing algorithm that explicitly accounts the demands of tasks based on their levels of confidence; and (ii) an Enhanced dual-mode scheduling policy to schedule tasks within a core. The proposed C-PEDF integrates every single high-level workload with a group of low-level workloads and coalesces them into a cluster. Within each cluster, tasks are scheduled under our Enhanced dual-mode scheduling policy to improve the service level of high-level tasks without jeopardizing the schedulability of low-level tasks. Clusters are scheduled under Earliest Deadline First (EDF) scheduling approach. We conduct a schedulability test for the proposed technique, and we demonstrate how workloads can be clustered by means of Mixed Integer Nonlinear Programming (MINLP) model. Extensive simulation results reveal that our algorithm significantly outperforms other existing approaches both in acceptance ratio and the impact factor of low-level tasks.

Keywords: clustering; mixed-criticality; multicore processor; task scheduling; schedulability; sporadic task; UAV.

1. Introduction

The relentless developments in microelectronic technology enable processor manufacturers to fabricate more computational elements (cores) on a single chip to realize high performance and better reliability at low-cost. Such integrated systems are efficiently used in numerous safety-related industrial sectors (e.g., automotive, medical, aerospace, home electronics market, nuclear power station, etc.,) to fulfil the radically increased computing demands of safety-critical applications. A system is said to be safety-critical whose malfunction might lead to loss of human life or serious damage to property/environment. Most of the safety-critical domains are mixed-critical which consolidate various functionalities with different criticality levels (i.e., importance) on a common computational platform. The adoption of multicore

processors for MCS will be driven by the increasing demand of computational power, development cost, and by SWaP (Space, Weight, and Power consumption) concerns.

An excellent example of MCS is the Unmanned Aerial Vehicles (UAV), generally known as pilotless aircrafts or drones [1]. Drone is a remotely operated airborne vehicle and finding applications in different private and public sectors ranging from military operations to traffic monitoring. It will have combat and surveillance abilities surpassing those of today's piloted airplanes. UAV combines tasks of different criticalities and executes them on a common embedded platform. The workloads (i.e., tasks) of UAV can be characterized into three categories:

1. *Flight-critical tasks:* High-level workloads that execute safety-related functionalities, such as flight control and trajectory computation to preserve the stability of aircraft, losing which a drone cannot be flown securely. The failures of these tasks can lead catastrophic consequences for the aircraft and hence need to be executed with the highest level of assurance.
2. *Mission-critical tasks:* Low-level tasks, which are concerned with surveillance objectives such as tracking potential targets, navigation services, and parking assistance, losing which a drone is still considered safe. Malfunction on these tasks can result in minor service disruption in the system that is not catastrophic.
3. *Non-critical tasks:* Tasks that execute the least important background activities. Task associated with vehicular entertainment, such as music streaming, is a good example of non-critical functionality that is least important for the specified mission.

Criticality is the degree of required protection against failure for a subsystem. It is called as Safety Integrity Levels (SIL) or Design Assurance Levels (DAL). These SILs reflect the required level of risk reduction in the engineering of a safety-related system and hence, influence all the phases of designing, implementing, testing, and certification processes. For example, in aeronautics standard DO-I78B, there are five SILs, characterized based on their level of jeopardizing produced by the failure of the task: *catastrophic; hazardous; major; minor; no effect* [2]. The task with higher criticality level indicates that a higher degree of guarantee is required for the correctness of system behaviour. For instance, in the control system of a UAV performing reconnaissance assignment, it is essential to assure the correctness of flight-critical activity such that the flight does not crash, than for mission-critical activities like sensing and transmitting images.

MCS often essential to be certified according to their criticality by a standard statutory third party, called as a Certification Authority (CA) (e.g., Federal Aviation Authority (FAA) in US and the European Aviation Safety Agency (EASA) in Europe for aerospace industry) [3]. Certification (i.e., conformity of assessment) is about guaranteeing different levels of the rigorous correctness of the system. To certify the correctness, CA mandates extremely rigorous and conservative assumptions regarding the run-time behaviour of the system, which are very unlikely to befall in reality. These authorities are not concerned about anything else apart from the safety of the aircraft. It is not incumbent on them whether the mission-critical functionalities are performed in time or not. Conversely, the entire system, comprising both the flight and mission-critical applications, must be authenticated by the system designers or other standard bodies, who generally use a considerably less rigorous standard than the one used by CAs.

Assimilating various operations on the same computational platform brings lots of upsides to the electronics market, enabling us to schedule more tasks hence maximizing the resource usage while reducing the SWaP requirements of the system. One of the major technical challenges in scheming MCS is formulating a scheduling criterion that exacerbates both the criticality and deadline (i.e., urgency) problems of the tasks while enabling certification. Moreover, it is essential to ensure sufficient separation and timeliness assurances for such MC workloads according to their safety and security levels. Any undesirable interference among these tasks must be prohibited to ensure the service guarantees of safety-related applications.

There is an extensive literature on scheduling MC tasks on both single-core and multicore systems. Unfortunately, traditional scheduling algorithms for MC tasks on multicore processors have at least one of the

following crucial limitations that severely restrict the practical applicability of those algorithms:

(i) Traditional scheduling approaches assume that if any high-criticality task shows its critical behaviour (i.e., it overruns its corresponding nominal low-level Worst-case Execution Time (WCET)), all currently active and upcoming critical tasks in the same processor are likely to show high-criticality behaviour, which is often impractical. Our scheduling algorithm eradicates this dispute by taking the behaviour of every high-level workload into account and leads to improved performance and schedulability.

(ii) Existing approaches cannot completely adapt the dynamic nature of the applications: the workload can dynamically fluctuate among high- and low-level execution modes.

(iii) Conventional scheduling algorithms do not deliver a good real-time guarantee for the low-level workloads: since these are performed opportunistically with unbounded termination to assure the timing guarantees for critical workloads. This is undesirable because low-level workloads also need some level of timeliness guarantees (even though with lower-level assurances than the high-criticality tasks).

Motivated by previous investigations on MC scheduling, we develop a new hierarchical scheduling framework, called Clustered-PEDF (C-PEDF) to schedule MC sporadic task systems that overcome the limitations posed by existing scheduling algorithms. The contributions of this paper are four-fold:

- We propose a Clustering-based partitioned MC scheduling strategy for implicit-deadline sporadic workloads on homogeneous (i.e., identical) multicore system, which imposes strong temporal separation between high-level workloads so they cannot influence each other and also provides better timeliness assurance for low-level workloads;
- We devise a C-PEDF scheduling approach which exploits a bin-packing algorithm for effective resource utilization that explicitly accounts the demands of tasks based on their levels of confidence and an Enhanced dual-mode scheduling policy to schedule tasks within a core;
- We develop a mathematical tool for schedulability analysis on multicore platforms under our C-PEDF scheduling approach. According to the schedulability constraints, we develop an evolutionary intelligence technique, namely MINLP model for task clustering, that can enhance the schedulability of the system;
- We evaluate our C-PEDF algorithm extensively against some existing partitioned scheduling algorithms.

Our article is structured as follows: The following Section provides considerable relevant algorithms aiming to schedule MC tasks on single and multicore platforms. The MC sporadic task model is formally defined in Section 3. Our Clustered partitioning MC scheduling strategy upon a single-core processor is discussed in Section 4. We formulate and solve an MINLP model to identify the scheduling constraints in Section 5. Enhanced Dual-mode Scheduling Mechanism is explained in Section 6. Clustering-based partitioned MC scheduling on a multicore processor is described in Section 7. We discuss experimental results in Section 8. We briefly describe our future work in Section 9. Finally, Section 10 concludes this paper.

2. Related Work

Scheduling tasks at different criticality levels on a multicore computational platform is the subject of an emerging research field due to the proliferation of safety-related real-time applications. A large body of MC task scheduling algorithms has already been proposed on single-core and multicore platforms. Conventionally, scheduling algorithms are categorized into two main paradigms: (i) global scheduling (ii) partitioned scheduling.

In a partitioned approach, workloads are always mapped to same cores (processing elements) and task migration is not allowed. This type of scheduler exploits distinct queues for each core and each task is mapped to one particular core a priori, and will only be scheduled on that core. Partitioned algorithms are often a favorable choice for hard real-time applications since it delivers a simple and more predictable operation [4]. The major disadvantage of this approach is inefficient resource utilization owing to assignment problems that arise in

the modeling of scheduling tasks to processing elements.

A global scheduler allows any task to execute on any core and migrates them across multiple executing cores. These approaches are often a better choice for soft real-time applications because the task-to-core mapping disputes that arise under partitioning approaches are mitigated if the limited delay is acceptable [5] [6]. A Global scheduler is most appropriate for workloads that are delivered based on the average execution time [7]. One key disadvantage of global scheduling is the interference across shared resources which make the WCET analysis more challenging.

The MC task scheduling problem was first identified and formalized by Vestal [8], in the scope of the static-priority preemptive scheduling. In his work, the schedulability is based on the WCET estimates of workloads of equal or greater priority. Vestal demonstrated that, from the viewpoint of low-criticality tasks, the WCET assigned for high-criticality tasks are unnecessarily pessimistic. Hence, the author suggested that schedulability tests for low-criticality workloads be dynamically adjusted to adapt less pessimistic WCET parameters for high-criticality workloads. This approach has been proved to be optimal by Dorin et al. [9]. Later, Baruah et al. extended Vestal's algorithm [8] to allocate static priorities on a per-job basis (instead of task level). They introduced an algorithm, named OCBP (Own Criticality-based Priority), to address the asymmetric effects among various criticality levels by exploiting more global knowledge of the system [10]. Such a global knowledge is much more effective than simple scheduling constraints like urgency or importance, and OCBP significantly outperforms other policies like EDF and Criticality Monotonic (more critical tasks have greater priority).

To explore the exact intricacy of the MC scheduling algorithms [10], Baruah et al. presented [11] two necessary schedulability conditions, the Worst-Case-Reservations (WCR) and OCBP. This work was extended by Li and Baruah [12] using a static-job-priority scheduling policy based on the OCBP condition, as well as in some following works [11], [13], [3]. All of the above approaches have focused on a single-core processor, but of late efforts have shifted in the direction of multicore platforms.

To our knowledge, the MC task scheduling on multicore platforms was first proposed by Li and Baruah [14]. Later, Gu et al. [15] extended the work on uniprocessor MC scheduling [16] to multicore systems and presented two improvements for processor schedulability by exploiting criticality-cognizance policy. Mollison et al. developed a scheduling heuristic for MC tasks on multicore platforms, implementing various methods (i.e., Cyclic Executive, partitioned-EDF, global-EDF and global best effort) for five different criticalities and enabling temporal isolation between tasks through a bandwidth reservation server [17]. In this work, the more critical tasks are executed with high priorities and the less critical tasks can run in the residual slack time.

Pellizzoni et al. developed a reservations-based scheme to enable sufficient separation between tasks with different criticality levels. They proposed an architectural technique for facilitating such isolation despite to allow some required interferences (e.g., in the sharing of non-preemptible system resources) among applications of various criticalities [18]. Their objective is not on enhancing processor utilization, but on guaranteeing isolation. Anderson et al. introduced a two-level hierarchical approach with a bandwidth reservation server to ensure timing isolation between tasks [19]. They also analyzed the importance of slack re-distribution methods to reallocate residual computing capacity at higher criticality levels to lower criticality levels.

Kelly et al. studied the issues of schedulability that arise in the conventional partitioning approaches under static-priority scheduling on multicore systems. They proposed an SIL-based WCET for the tasks and a dispatching algorithm for mapping tasks to processor [20]. In order to explore both general and criticality mode change protocols, Burns surveyed the pessimistic assumptions of high criticality tasks in the higher criticality modes [21]. Petters et al. analyzed the significance of timing isolation of tasks for MCS and explored several problems in designing such systems in practice [22]. The disadvantage of the timing isolation method is that it operates on rigorously over-provisioning system resources, which may cause an adverse impact on the resource allocation cost and power performance of the system.

In contrast to these approaches, our scheduling algorithm can provide isolation between criticality levels from the perception of temporal correctness. The propagation of high-level execution behavior of a

particular task is prohibited by enabling strong isolation between various clusters. In order to maximize the total schedulable utilization, our algorithm allows the system to transit back from high- to low-level execution mode as soon as all critical workloads show their low-level execution behavior.

3. System Model and Notation

In this section, we formally define the MCS task model and describe terms and notions used in this article. Since our eventual interest is in the scheduling of finite collections of recurrent (periodic or sporadic) MC tasks, we will consider the scheduling instance contains a finite set of sporadic tasks upon a homogeneous multicore platform. For simplicity, we confine our attention to a dual-criticality system. The high-critical tasks must finish their execution before their next release time. We relax the timeliness constraints for low-critical tasks in that some level of deadline misses are tolerated. In a dual-criticality task system, each sporadic task is defined as $\tau_i = (P_i, D_i, £_i, C_i^1, C_i^2)$ with the following semantics:

- $i \in N^+$ is a unique index of task (i.e., $1 \leq i \leq n$)
- $P_i \in R^+$ is the minimum inter-arrival time (period) of workload τ_i.
- $D_i \in R^+$ is the completion deadline of task τ_i. We consider an implicit-deadline sporadic task system in which each task τ_i has a deadline D_i equal to its period P_i. (i.e., $\forall \tau_i \in \tau$, $P_i = D_i$), meaning that each task must be finished before its next release time.
- $£_i \in \{1, 2\}$ specifies the criticality of the task τ_i. Tasks with $£ \leftarrow 1$ and $£ \leftarrow 2$ designating low- and high-criticality level respectively.
- $C_i^1 \in N^+$ specifies the WCET of τ_i at the low-criticality level.
- $C_i^2 \in N^+$ specifies the WCET of τ_i at the high-criticality level.

Every high-criticality task τ_i^2 is represented by two WCET values: (i) more pessimistic, high-level WCET C_i^2 considered by CA; and (ii) a less pessimistic, low-level WCET C_i^1 expected by the system engineer. This is because the high-level WCET is extremely more pessimistic than low-level WCET (to ensure timeliness guarantee). Therefore, for each high-criticality task $C_i^2 \geq C_i^1$. Each low-level task τ_i^1 is characterized by a single low-level WCET C_i^1. We also consider that the low-criticality tasks are enforced to suspend after C_i^1 time units of execution. Tasks are not permitted to migrate across cores after assignment as we consider partitioned scheduling policy. The notations used to describe the proposed scheduling mechanism are listed in Table 1.

Table 1. Notations

τ_i	Task i
τ_i^1	Low-criticality task
τ_i^2	High-criticality task
P_i	Minimum inter-arrival time of workload τ_i
D_i	Deadline of task τ_i
$£_i$	Criticality of the task τ_i
C_i^1	WCET of τ_i at the low-criticality level
C_i^2	WCET of τ_i at the high-criticality level
U_i^1	Task utilization under low- criticality mode
U_i^2	Task utilization under high-criticality mode
S_j	Cluster j
P_{Sj}	Base period of the cluster S_j
LO_i^j	Number of cluster budget replenishments during $P(\tau_i^1)$
HI_j	Number of cluster budget replenishments during $P(\tau_i^2)$
E_j	Execution time budget

Example 1: Consider a simple dual-criticality task set comprised of three sporadic implicit-deadline tasks τ_1^2, τ_1^1, and τ_2^1. The task parameters are given in Table.2 with time information in units of ms.

Table 2. An example task set

Task	£	C_i^1	C_i^2	$P_i(\tau_i)$
τ_1^2	2	3	12	15
τ_1^1	1	4	-	10
τ_2^1	1	3	-	15

Assume the first jobs of all the tasks have arrived at time zero. Task τ_1^2 is flight-critical which has criticality level 2 (higher-criticality level) and other two tasks τ_1^1 and τ_2^1 are only mission-critical with criticality level 1 (lower-criticality level). C_i^1 defines low-level WCET and C_i^2 defines the high-level WCET.

3.1. Scheduling Assurance

The given set of MC tasks is said to be schedulable if and only if the conditions associated with dynamic timing properties for all the tasks are satisfied:

- *Low-level assurance*: If all tasks do not overrun their low-level WCET C_i^1, all high- and low-criticality tasks are assured to receive enough execution to satisfy their deadlines.
- *High-level assurance*: If all workloads do not overrun their high-level WCET C_i^2 and at least one high-level workload overruns it's low-level WCET C_i^1, all high-level workloads are assured to satisfy their timing constraints (whereas low-criticality tasks may be suspended).

3.2. Utilization

The utilization of a workload is usually defined as the fraction of its worst case execution time to inter-arrival time. Hence, the utilization of MC sporadic task τ_i under low and high-criticality modes is defined as follows:

$$U_i^1(\tau_i) = \frac{C_i^1}{P_i} \quad (1)$$

$$U_i^2(\tau_i) = \frac{C_i^2}{P_i} \quad (2)$$

where $U_i^1(\tau_i)$ and $U_i^2(\tau_i)$ specify the task utilization under low- and high-criticality mode, respectively. For our example task set depicted in Table 2, $U_i^1(\tau_i)$ and $U_i^2(\tau_i)$ are computed for each task. The same cluster is given in Table 3 with an extra column for their utilization.

Table 3. Task set with its utilization parameters

Task	Given parameters				Calculated parameters	
	£	C_i^1	C_i^2	$P_i(\tau_i)$	$U_1^1(\tau_i)$	$U_1^2(\tau_i)$
τ_1^2	2	3	12	15	0.2	0.8
τ_1^1	1	4	-	10	0.4	-
τ_2^1	1	3	-	15	0.2	-

The cumulative utilization of all low-level tasks under low-criticality mode $U_i^1(\tau)$ and the cumulative utilization of all high-level tasks under high-criticality mode $U_i^2(\tau)$ are calculated as follows:

$$U_i^1(\tau) = \sum_{\tau_i \in \tau \wedge £=1} U_i^1(\tau_i) \quad (3)$$

$$U_i^2(\tau) = \sum_{\tau_i \in \tau \wedge £=2} U_i^2(\tau_i) \quad (4)$$

Theorem 1 [23]**:** *A dual-criticality sporadic task set* $\tau_i = \{\tau_1^£, \tau_2^£, \ldots, \tau_n^£\}$ *is schedulable under EDF approach over a (core) processor if:*

$$U_i^1(\tau) + U_i^2(\tau) \leq 1 \quad (5)$$

We would like to point out that the condition given in Eq. (5) hinges only on the upper bound processing capacity of high-level workloads and the lower bound processing capacity of low-level workloads. From this observation, it is clear that determining the optimal partitioning of admitted workloads to hold the inequality (5) is NP-hard [24]. Now, we examine the schedulability of the admitted workloads depicted in Table 3 against the system utilization. Since the cumulative utilization of the tasks at their own criticality levels is (0.8+0.4+0.2) =1.4 ≥ 1, the task system cannot be schedulable on a single-core processor under high-criticality mode.

4. Clustering-based MC scheduling on single-core Processor

In order to impose strong isolation among high-level workloads and to decrease their service intervention with low-level workloads, each high-level workload τ_j^2 is scheduled with a group of low-level workloads and coalesces both in a distinct cluster. In our scheduling architecture, tasks within each cluster are scheduled under a budget-driven scheduling mechanism, and clusters are scheduled using EDF strategy.

4.1. Clusters

We now define the task clusters. A cluster is a collection of tasks gathered together before each task is scheduled. In our scheduling mechanism, every cluster comprises of one executive (high-level) task and a group of member (low-level) tasks. The cluster is denoted by $S_j = \{\tau_j^2, \tau_1^1, \tau_2^1, \tau_3^1, \tau_4^1, \ldots, \tau_n^1\}$ where the τ_j^2 ($1 \leq j \leq m$) is the single executive task and tasks τ_i^1 ($1 \leq i \leq n$) are member tasks. The base period (P_{Sj}) of the cluster S_j is computed as the greatest common factor (gcf) of inter-arrival time of all workloads in the particular cluster, i.e.,

$$P_{S_j} = gcf\left(P(\tau_j^2), P(\tau_1^1), P(\tau_2^1), \ldots P(\tau_n^1)\right) \quad (6)$$

where $P(\tau_j^2)$ is the period of the executive task τ_j^2 and $P(\tau_i^1)$ is the period of member tasks τ_i^1 $(1 \le i \le n)$. For simplicity, the period of the cluster specified as S-period. The number of cluster budget replenishments during $P(\tau_i^1)$ is calculated as

$$LO_i^j = \frac{P(\tau_i^1)}{P_{S_j}} \quad , \quad (1 \le i \le n) \qquad (7)$$

Similarly, the number of budget replenishments during $P(\tau_j^2)$ is calculated as

$$HI_j = \frac{P(\tau_j^2)}{P_{S_j}} \quad , \quad (1 \le j \le m) \qquad (8)$$

The E_j denotes the execution time budget that a cluster S_j must collect to guarantee all of its workloads fulfil the MC schedulability condition. In C-PEDF technique, each cluster S_j will be scheduled with other clusters as a normal sporadic task with period P_{Sj} and execution time budget E_j. As stated in the previous section, we define the utilization of a cluster to be E_j / P_{Sj}.

4.2. Mode Transition Protocol

The basic idea of the runtime mechanism used in our scheduling strategy is the operation of mode transition protocol. The scheduling process within each cluster S_j is achieved in cycles; each cycle corresponds to a period of the executive task τ_j^2 (i.e., P (τ_j^2)). Within each cycle, there can be three distinct behaviours. These three run-time behaviours seem to reveal three different operating modes, which are shown in Fig.1.

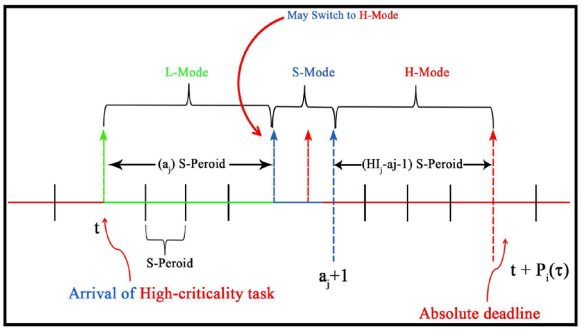

Fig.1. Mode transition strategy in a cluster

(i) *Low-criticality behaviour* (L-Mode): The system starts it's execution with this normal mode, where currently-active job (arrived, but not yet finished) of task τ_j^2 has not yet reached its respective C_i^1 and hence it is in L-Mode. The entire tasks in the cluster are executed to provide the low-level assurance.

(ii) *Transition or switching behaviour* (S-Mode): If any active job of τ_j^2 overruns it's low-level WCET, then the system switches immediately to high-level execution behaviour during the active S-period. All the member tasks in the cluster are cancelled to provide the high-level assurance.

(iii) *High-criticality behaviour* (H-Mode): In this mode, an active job of τ_j^2 has either completed or executed beyond its C_i^1, so it is known whether the cluster is in L-Mode or H-Mode. If the executive task has really completed and the cluster has not yet depleted its reserved execution budget, any suspended member tasks will remain serviced; but, if they do not finish by the last S-period of every cycle (whereas another executive task arrives) or by their deadlines, whichever is earlier, they likely to be discarded. If the cluster is in H-Mode, the member tasks continue to be suspended. If at least one executive task overruns its corresponding high-level WCET C_i^2, then the scenario is called erroneous.

4.3. Three-mode Budget-driven Scheduling within a Cluster

We now provide the execution semantics of our mechanism, describing what happens during each cycle of the scheduling process. Consider the allocated execution budget of the cluster S_j is E_j and the number of execution replenishments in any $P(\tau_j^2)$ is HI_j. As shown in Fig.1, we assume that L-Mode lasts for a_j S-periods, that is, the active job of τ_j^2 either changes to the H-Mode or has completed in $(a_j + 1)$th S-periods. In every cycle, tasks are executed according to their predefined budget values, t_j, $e1_i$ and $e2_i$ (i.e., i = 1,2,3…n). The technique for estimating these constraints is explained in the next section.

L-Mode ([0, a_j] S-periods): In order to realize the low-level assurance, in each [0, a_j] S-periods, workloads are executed as: if there a``re any incomplete member tasks, the active job of τ_j^2 is performed first for a definite amount of time $t_j \ge 0$ (as a maximum). As soon as this job finishes, member tasks are performed in a non-decreasing order of the task index, where all τ_i^1 is allotted with an execution budget of $e1_i \ge 0$. If there are no incomplete member tasks, the active job of τ_j^2 is performed until it finishes its execution or the reserved execution time of the particular cluster is exhausted.

The pre-allocated budget parameters should hold the following conditions:

(i) The entire execution budget assigned to all workloads in the cluster should not overrun the reserved execution budget E_j of the cluster. Hence,

$$E_j \geq t_j + \sum_{i=1}^{n} el_i \quad (9)$$

(ii) As the active job of the executive task exhibits its normal behaviour, the execution time allocated to τ_j^2 in this L-Mode should not overrun its low-level WCET, i.e.,

$$C^1_{\tau_j^2} \geq a_j \times t_j \quad (10)$$

S-Mode ($[a_j, (a_j+1)]$ S-period): In order to realize the high-level assurance, the active job of τ_j^2 is performed first with an allocated execution time t_j. If this executive job finishes its execution, member tasks are executed until E_j is depleted (whereas some member jobs may be suspended). Else, the system will change from low- to high-level execution mode and will remain performed until it finishes or E_j is depleted.

As stated above, condition (9) should hold to guarantee the budget of t_j for executive task τ_j^2 and the budget $e1_i$ for each member task τ_i^1. Furthermore, since the active job of τ_j^2 will change to the H-Mode if it cannot finish with t_j, the budget expected by τ_j^2 before the behaviour change should be greater than or equal to its corresponding low-level WCET. Hence,

$$(a_j + 1) \times t_j \geq C^1_{\tau_j^2} \quad (11)$$

H-Mode ($[(a_j+1), HI_j]$ S-periods): In order to provide the high-level assurance, in each of the last $(HI_j - a_j - 1)$ S-periods, tasks are executed as follows: if the active job of τ_j^2 has not finished in the $(HI_j - a_j - 1)$ S-period, it will be performed until it finishes its execution or the received budget E_j is depleted. Else, member tasks are serviced by non-decreasing task index, where each τ_i^1 is assigned an execution time of $e2_i \geq 0$. Since the entire budget E_j will be allotted to member tasks, we can assign more budget to each member. i.e.

$$0 \leq e1_i \leq e2_i \quad (12)$$

At the same time, the budget allotted to the entire low-level workloads should not overrun the reserved execution time of the cluster:

$$E_j \geq \sum_{i=1}^{n} e2_i \quad (13)$$

We now demonstrate our scheduling strategy using following example.

Example 2: Consider the cluster comprised of three tasks in the Table 3. The base period of the cluster can be calculated as $P_{S1} = gcf\,(15, 10, 15) = 5$. Assume the execution time allotted to S_1 is $E_1 = 4.25$ and other scheduling constraints are $a_1 = 0$, $t_1 = 3$, $e1_1 = 1.25$, $e2_1 = 2.75$, $e1_2 = 0$ and $e2_2 = 1.5$. Fig.2 exemplifies the L-Mode behaviour of the executive task, whereas in Fig.3 it shows its H-Mode behaviour. Each cycle covers $HI_j = P(\tau_1^2) / P_{S1} = 15 / 5 = 3$ S-periods (i.e., 15ms) and involves only L-Mode and H-Mode (since $a_1 = 0$).

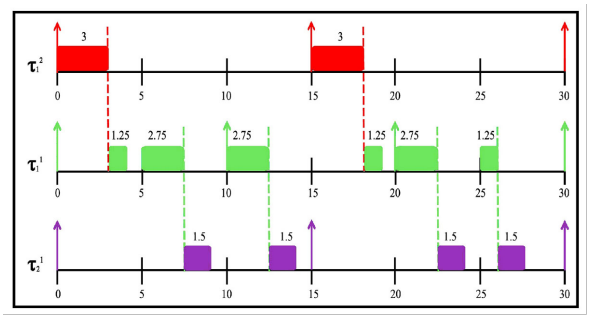

Fig.2. Scheduling of the task set under L-Mode.

In figure 2, for the first S-period (i.e., [0, 5]), we can see the executive task τ_1^2 is first performed for $t_1 = 3$ and finishes its execution; then, the member task τ_1^1 is performed for the execution budget $e1_1 = 1.25$ (τ_2^1 is not performed at this interval since $e1_2 = 0$).

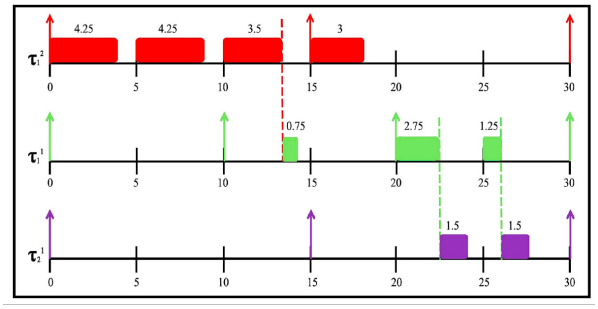

Fig.3. Scheduling of the task set under H-Mode.

Since the cluster is in L-Mode, both member tasks τ_1^1 and τ_2^1 are performed in the second S-period ([5, 10]) and third S-period ([10, 15]) for $e2_1 = 2.75$ and $e2_2$

= 1.5 correspondingly. The second scheduling cycle ([15, 30]) works in the same way, but τ_1^1 completes before depleting its entire allotted budget in the time interval [25, 30].

In Fig.3, τ_1^2 overruns its low-level WCET in the first cycle ([0, 15]), so the cluster is changed immediately to high-level execution behaviour, and all member tasks are cancelled (till τ_1^2 finishes). In the interval ([15, 30]), τ_1^2 continues in L-Mode, therefore τ_1^1 and τ_2^1 are performed normally, according to their allotted xecution time. In the first S-period of this second cycle, (i.e., [15, 20]), neither τ_1^1 nor τ_2^1 is performed. The reason is, the currently-active task τ_1^1 has been suspended at time 15ms when next τ_1^2 arrives, whereas the budget allotted to τ_2^1 is $e1_2 = 0$. It is important to note that only few member tasks violate their timing constraints: member tasks arrived in [0, 15] are discarded, but the entire member tasks arrived in [0, 15] are schedulable.

Let us consider a cluster $S_j = \{\tau_j^2, \tau_1^1, \tau_2^1, ..., \tau_n^1\}$. Here, we first develop the MC schedulability constraints for S_j under our mode transition protocol with a pre-allocated execution time E_j and other constraints a_j, t_j, $e1_i$, $e2_i$ ($1 \le i \le n$), where $t_j \ge 0$, $e1_i \ge 0$, $e2_i \ge 0$ and $a_j > 0$. We then identify the sufficient schedulability condition for a cluster upon a single-core system. Lemma 1 gives the schedulability conditions for every member task τ_i^1.

Lemma 1: *All the member tasks τ_i^1 ($1 \le i \le n$) are guaranteed to be schedulable if and only if all the conditions (9) – (14) satisfy, where the condition (14) is given below:*

$$N_i^{OL} \times e1_i + \left(LO_i^j - N_i^{OL}\right) \times e2_i \ge C_{\tau_i}^1 \qquad (14)$$

where N_i^{OL} is the upper bound of S-periods in $P(\tau_i^1)$ that overlap with L-Mode and S-Mode of a scheduling cycle. It is calculated as follows:

$$N_i^{OL} = \left\lfloor \frac{LO_i^j}{HI_j} \right\rfloor \times (a_j + 1) + \min\left\{LO_i^j \bmod HI_j, a_j + 1\right\} \qquad (15)$$

Example 3: Let us use an example given in Table 4, with pre-allocated execution time $E_1 = 1.5$, $a_1 = 1$, $e1_1 = 0$ and $e2_1 = 1.5$ to exemplify the lemma 1. The estimated value of other scheduling parameters are $P_{S1} = 5$, $LO_i^j = 3$, and $HI_j = 5$. The upper bound of the number of overlapping S-periods during is calculated as follows:

$$N_i^{OL} = \left\lfloor \frac{3}{5} \right\rfloor \times (1+1) + \min\{3 \bmod 5, 1+1\} \quad = 2$$

Table 4. An example cluster with task parameters

Task	Given parameters				Calculated parameters	
	£	C_i^1	C_i^2	$P_i(\tau_i)$	$U_1^1(\tau_i)$	$U_1^2(\tau_i)$
τ_1^2	2	3	7.5	25	0.12	0.3
τ_1^1	1	1.5	-	15	0.10	-

There are 2 overlapping S-periods for the first and fourth arrivals of τ_1^1, 1 for the second and third arrivals, and zero for the fifth arrival as illustrated in Fig. 4. Now, we can certify that all jobs of τ_1^1 can complete before their deadlines.

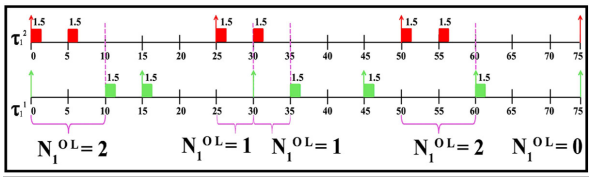

Fig.4. The number of overlapping S-periods

Lemma 2: *Each executive task τ_j^2 collects an execution budget of at least $C_{\tau_j^2}^1$ when it is in L-Mode and at least $C_{\tau_j^2}^2$ when it is in H-Mode by its deadline if the conditions (9), (11) and the following condition (16) satisfy:*

$$C_{\tau_j^2}^2 \le a_j \times t_j + (HI_j - a_j) \times E_j \qquad (16)$$

From lemma 1 and 2, we can drive theorem 2.

Theorem 2: *Suppose each cluster $S_j = \{\tau_j^2, \tau_1^1, \tau_2^1, \tau_3^1, ..., \tau_n^1\}$ is assured to collect an execution budget of E_j in each of its periods. Then, the cluster is MC-schedulable with constraints E_j, a_j, t_j, $e1_i$, $e2_i$ ($1 \le i \le n$) if all of the conditions (9) – (16) are met.*

Let G_λ be the group of clusters scheduled on a core λ. Note that each cluster $S_j \in G_\lambda$ is scheduled with other clusters in G_λ as a normal sporadic task with period P_{Sj} and budget E_j using EDF policy. The cumulative total processing capacity of the group of clusters is denoted by U_λ. According to the schedulable demand-bound

functions of EDF [24], all clusters in G_λ are able to schedule if:

$$U_\lambda = \sum_{s_j \in G_\lambda} \frac{E_j}{P_{s_j}} \leq 1 \qquad (17)$$

Hence, every cluster S_j is received an execution budget of E_j in each P_{Sj} if $U_\lambda \leq 1$. From the above inequality (17) and Theorem 2, now we are ready to derive the sufficient condition and establish our main theorem for clusters schedulability.

Theorem 3: *Let G_λ be the group of clusters scheduled on the core Ψ_λ (based on any task clustering approach). If for all $S_j = \{\tau_j^2, \tau_1^1, \tau_2^1, \tau_3^1, \tau_4^1, \ldots, \tau_n^1\}$ in G_λ, there exist value $E_j \in R^+$ and constraints $a_j \in N$ ($a_j < HI_j$) and t_j, $e1_i$, $e2_i \in R^+$ where $1 \leq i \leq n$ such that $U_\lambda \leq 1$ and the conditions (9) – (16) are met, then every cluster mapped to the core λ are MC schedulable.*

5. Mixed Integer Nonlinear Programming Model for Task Clustering

Fig.5 illustrates a unified overview of our proposed mechanism. Consider a core Ψ_λ in which the set of tasks $\tau(\Psi_\lambda) = (\tau_1^2, \ldots\ldots \tau_m^2, \tau_1^1, \ldots, \tau_n^1)$ are scheduled. Each task has its own parameters P_i, D_i, £$_i$, C_i^1 and C_i^2. Our objective of the task clustering is to find (i) a group of the member tasks that will be clustered with executive task τ_j^2 ($1 \leq j \leq m$) into the cluster S_j; and (ii) for every cluster S_j, the values of the execution budget E_j and the other constraints a_j, t_j, $e1_i$, $e2_i$, P_{Sj}, LO_i^j and HI_j.

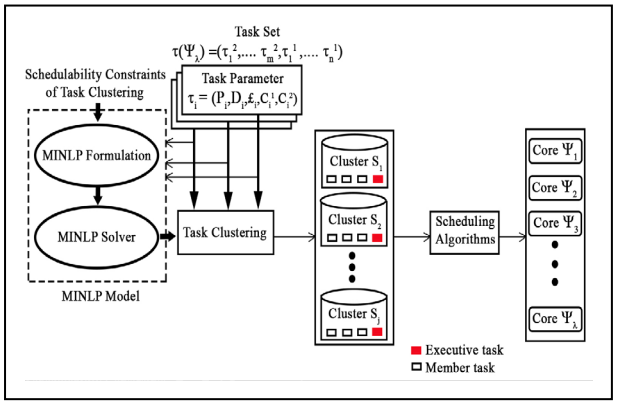

Fig.5 Overview of proposed mechanism

Here, we develop an MINLP model for clustering according to the schedulability constraints. Since the clusters are executed using EDF policy with periods P_{Sj} and a pre-allocated execution time E_j, the smaller utilization indicates better schedulability. Hence, the objective function of the MINLP problem is to minimize the overall utilization of the clusters so as to enhance their percentage of schedulable tasks. The parameters of the MINLP are identified from the MC schedulability constraints of the clusters. However, the MINLP is of NP-hard complexity and therefore computationally expensive to be solved every time in order to determine the optimal solution.

In order to allow a member task to exploit the overprovisioning of more than one executive task, we enable each τ_i^1 to present in multiple clusters – that is, every τ_i^1 can obtain pre-allocated execution time from multiple clusters and the τ_i^1 is performed whenever it is organized in a cluster. In the meantime, all the clusters are serviced serially on a common computing platform; τ_i^1 is not ever performed concurrently by more than one cluster. Therefore, if the reserved execution time that the low-level task is assured to collect from all of its clusters is at least C_i^1, then τ_i^1 is schedulable.

According to the above tenet, new variables ($ej1_i$, $ej2_i$) are used to specify the predefined execution times that τ_i^1 can collect from cluster S_j in each scheduling modes of S_j (rather than using an only one set of values ($e1_i$, $e2_i$)). These variables are also used to specify whether the member task has its place in the cluster S_j: if $ej1_i = ej2_i = 0$, then τ_i^1 does not belong to S_j, or else it does. According to the Theorem 3, the task set $\tau(\Psi_\lambda)$ is MC schedulable if all of the conditions (9) – (16) are satisfied for each cluster S_j. Hence, the scheduling parameters for each cluster S_j can be calculated by replacing $e1_i$ with $ej1_i$, $e2_i$ with $ej2_i$, N_i^{OL} with $N_{i,j}^{OL}$ in the conditions and rewriting the condition (14) as

$$\sum_{j=1}^{m} \left(ej1_i \times N_{i,j}^{OL} + ej2_i \times \left(LO_j^i - N_{i,j}^{OL} \right) \right) \geq C_{\tau_i}^1 \qquad (18)$$

In order to decrease the processing intractability, rather than using the condition (9), we use $E_j = t_j + \sum_{i=1}^{n} ej1_i$.

Now, we can summarize a complete MINLP formulation for our utilization minimization problem in Fig.6. Putting together the objective function with the constraints, we define the MINLP model for our scheduling mechanism. According to the schedulability restraints, the MINLP model finds which tasks can be clustered to minimize the utilization. Given a set of

mixed-critical workloads, if the overall cluster utilization derived from the formulated mathematical model is greater than 1, then this task set is considered to be not schedulable by the algorithm and will be precluded by the scheduler.

Aiming to optimize the solution of the assignment, the MINLP problem is solved by a branch-and-bound solver, MINOTAUR (Mixed-Integer Nonconvex Optimization Toolbox: Algorithms, Underestimators, and Relaxations) [25, 26]. It is one of the best open-source solver toolkits which offer methods and data structures to study and solve complex MINLPs directly.

Minimize $\sum_{j=1}^{m}((\sum_{i=1}^{n} ej1_i + t_j)/P_{s_j})$

Subjected to

$E_j \geq t_j + \sum_{i=1}^{n} ej1_i$

$C^1_{\tau_j^2} \geq a_j \times t_j$

$(a_j + 1) \times t_j \geq C^1_{\tau_j^2}$

$0 \leq ej1_i \leq ej2_i$

$E_j \geq \sum_{i=1}^{n} ej2_i$

$\sum_{j=1}^{m} \left(ej1_i \times N^{OL}_{i,j} + ej2_i \times \left(LO^i_j - N^{OL}_{i,j}\right)\right) \geq C^1_{\tau_i^1}$

$a_j \times t_j + (HI_j - a_j) \times E_j \geq C^2_{\tau_j^2}$

$0 \leq a_j \leq HI_j - 1$

$a_j \in R^+$

$ej1_i \in R^+$

$ej2_i \in R^+$

$t_j \in N$

$\forall \tau_i \in \tau$ and $1 \leq i \leq n$, $\forall \tau_j \in \tau$ and $1 \leq j \leq = m$)

Fig.6. Scheduling constraints for MINLP formulation

MINOTAUR provides a number of algorithms and advanced routines/libraries to find the solution of the complex MINLPs and can be directly called from several other codes like AMPL (A Mathematical Programming Language) scripts [27], FORTRAN or C++. In order to solve MINLP mathematical model, MINOTAUR exploits LP/NLP-based branch-and-bound (LP/NLP-BB) algorithm [28]. The non-negative parameters a_j, t_j, $ej1_i$, $ej2_i$ and t_j indicates that the MINLP problem is convex, which guarantees that the solver provides an optimal result of the given mathematical model. The LP/NLP-BB solver is started by relaxing the linearization of the original MINLP problem and building a relaxed mixed-integer linear programming (MILP). The nonlinear constraint of an MINLP problem is represented by the following equation.

$$g(z) \leq 0 \qquad (19)$$

where $g(\cdot)$ is a convex function and z is the continuous variable. The integrality restriction can be relaxed by finding the linear approximation of the function about any point z^k as follows:

$$\nabla g(z^k)^T (z - z^k) + g(z^k) \leq 0 \qquad (20)$$

The MILP relaxation provides a close approximation to the original MINLP problem if we find the linear approximation about more points. On the other hand, the number of integrality constraints in the MILP problem increases with the number of linearization points and obviously reduces the speed of operation of MINOTAUR. To circumvent this issue, linearization limits obtained from a single point are included in the initial iteration. This initial estimate is considered as the initial solution of the relaxed nonlinear programming (NLP) [28]. Then, we take linearization for more points at which the constraints violated significantly. The LP/NLP-BB solver initiates by finding the solution for a linear programming (LP) relaxation and sets the incumbent value INC_V = ∞ [28]. The solver then generates a search tree to resolve very tight MILP relaxations. In each iteration, we exclude an LP subset from the list and resolve it. If the obtained result is more than the current INC_V, we reject this subset since it does not encompass any value superior than the INC_V. If the result $\hat{z}$ obtained from the linear programming subset has a fractional value, two additional subsets are created by subdividing (branching) the noninteger variable. Afterwards, these two newly created subsets are included in the list of unresolved subsets. If $\hat{z}$ holds every integer constraints, then we inspect whether it holds the nonlinear constraints or not. If it is feasible, we have a new INC_V. Otherwise, we successively linearize one or more perturbed constraints about $\hat{z}$ and continue. When there are no more remaining subsets left to resolve the algorithm will terminate.

In our problem formulation, the MINLP has m integer variables, 2mn+6m constraints, and m+2mn real variables, where m and n are the number of member and

executive tasks, correspondingly. Note that depending upon the size of the input (the size of task set), the amount of variables and constraints associated with the problem is varied. In the worst case, the solver needs to solve an exponential amount of linear and nonlinear subsets. Nevertheless, in reality, MINOTAUR produces significantly higher quality solutions in much less time. The mathematical model shown in Fig.6 can easily be reformed by accounting other kinds of constraints also – e.g., restrictions on cluster count that each member can fit, restrictions on the number of member tasks in each cluster, etc.

6. Enhanced Dual-mode Scheduling Algorithm

Unfortunately, the Three-mode budget-driven scheduling strategy introduces excessive overhead and unnecessary context switching when there are more member tasks in a cluster (refer Fig. 7). In order to evade this problem, we develop an Enhanced dual-mode (E-Mode) algorithm that schedules the member tasks using EDF policy with reduced switching overhead. Given a cluster S_j with allocated execution time E_j, a_j and t_j, the E-Mode mechanism operates as follows:

Mode I ([0, (HI_j−1)] S-periods): For every S-period, if there are some incomplete member tasks in the cluster, the active job of τ_j^2 is performed first for a certain amount of time (up to t_j), and then the member tasks in the cluster are performed using EDF approach. Else, the active job of τ_j^2 is performed until it finishes or E_j exhausts.

Mode II (the last ($HI_j - a_j$) S-period): For every S-period, the job of τ_j^2 is performed first until it completes and then the incomplete member tasks in the cluster are performed using EDF strategy. If the active job of τ_j^2 is in H-Mode, the member tasks that cannot complete when their deadlines are met or another τ_j^2 arrives are discarded. In contrast to the Three-mode scheduling mechanism, our E-Mode strategy considers (a_j +1)th S-period and the last ($HI_j - a_j$) S-periods as a single mode. The rationale behind this is that under the Three-mode mechanism, different execution budgets ($e1_i$ and $e2_i$) may be applied to each τ_i^1 in (a_j +1)th S-period and in ($HI_j - a_j$ −1) S-periods, whereas under the E-Mode policy, member tasks are always executed using EDF approach.

Theorem 4: *If a cluster* $S_j = \{\tau_j^2, \tau_1^1, \tau_2^1, \tau_3^1, \tau_4^1, ..., \tau_n^1\}$ *is MC schedulable under the Three-mode policy with an execution budget* E_j *and the constraints* a_j, t_j, $e1_i$, $e2_i$, *then* S_j *is also MC-schedulable under the E-Mode scheduling policy with the same budget* E_j *and scheduling constraints* a_j *and* t_j.

Example 3: Consider the cluster given in Table 5, with the parameters $E_1 = 4.5$, $a_1 = 0$ and $t_1 = 3$. Each member task is scheduled under the Three-mode scheduling mechanism with execution budgets $e1_1$ =1.25, $e2_1$ =2.75, $e1_2$ = 0.25 and $e2_2$ = 1.75 as depicted in Fig.7. The same task set is scheduled under E-Mode scheduling strategy as shown in Fig.8. E-Mode reduces the number of mode transitions, and thus decreases the context overheads.

Table 5. An example cluster with task parameters

Task	Given parameters				Calculated parameters	
	£	C_i^1	C_i^2	$P_i(\tau_i)$	$U_i^1(\tau_i)$	$U_i^2(\tau_i)$
τ_1^2	2	3	13.5	15	0.2	0.9
τ_1^1	1	4	-	10	0.4	-
τ_2^1	1	3.75	-	15	0.25	-

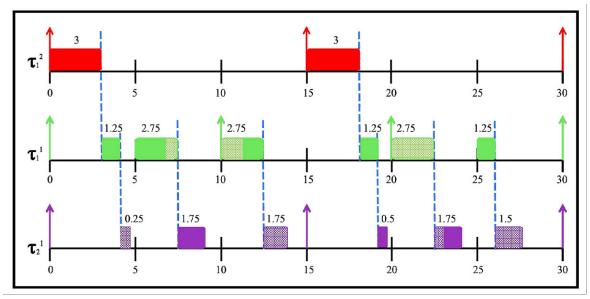

Fig.7. Three -mode Budget-driven schedule

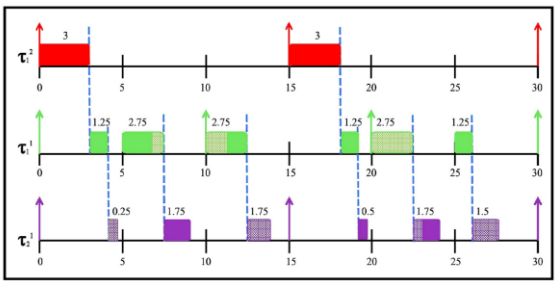

Fig.8. Schedule using enhanced dual-mode

From Theorem 4, we can estimate the constraints for E-Mode mechanism by resolving the MINLP model formulated in Section 5. E-Mode can alleviate the switching overhead by dividing the number of low-level workloads as a main constraint. Furthermore, we can also provide a timing assurance for each τ_i^1 providing a restriction on the number of clusters in the system.

7. Clustering-based Partitioned MC scheduling on a Multicore Processor

We now briefly discuss our Clustering-based partitioning EDF algorithm for dispatching admitted workloads to multiple cores. Let us denote a processor Ψ with λ homogeneous cores as $\Psi = \{\Psi_1,\ldots.\Psi_j\ldots,\Psi_\lambda\}$. Consider the group of tasks assigned to each core Ψ_j is represented by $\tau(\Psi_j)$. When the number of on-chip core increases, C-PEDF can satisfy the new requirements.

Our proposed approach targets to distribute the tasks among all cores while choosing an executive (member) task to allocate based on decreasing executive (member) task utilization. Consider $\tau^2 = \{\tau_1^2,...,\tau_{n1}^2\}$ be the group of executive tasks and $\tau^1 = \{\tau^1_{n1+1}\ldots\ldots, \tau^1_n\}$ be the group of member tasks. The formal steps of C-PEDF for scheduling tasks with dual criticality levels are given in Algorithm 1.

C-PEDF starts by assigning $\tau(\Psi_j)$ to null and by ordering the given set of workloads. Executive tasks are arranged in descending order of their $U_i^2(\tau)$, and tasks with equal $U_i^2(\tau)$ are again arranged in decreasing order of $U_i^1(\tau)$. Member tasks are arranged in descending order of $U_i^1(\tau)$, and workloads with equal $U_i^1(\tau)$ are again arranged in ascending order of their inter-arrival time. The motivation behind this ordering is: for executive tasks with equal $U_i^2(\tau)$, a task with a higher value of $U_i^1(\tau)$ typically has less overprovisioning; and for member tasks with equal $U_i^1(\tau)$, the workload with a lesser value of inter-arrival time will likely have a minimum utilization rate of overprovisioning, since P_{Sj} is selected as the *gcf* of the inter-arrival time of all the workloads.

Our C-PEDF then chooses the task (τ_s) with the highest utilization. Then, the algorithm selects an appropriate core Ψ_j for this workload, such that the cumulative utilization for selected workload in conjunction with the active workloads on this core is minimal; hence, it distributes the tasks across cores. If the cumulative cluster utilization on the chosen core Ψ_j does not go beyond one, τ_s is allocated to Ψ_j. Or else, the C-PEDF terminates with a failure report. If the workload is fruitfully assigned to a core, the C-PEDF reports "success". According to the Theorem 4, the scheduling achieved by our C-PEDF always guarantees the real-time performance of the admitted tasks. The equally critical workloads on a core are directly scheduled under EDF policy.

Algorithm 1: Pseudo code for Clustered-PEDF

Input: An MC task system with a group of executive tasks $\tau^2 = \{\tau_1^2,..., \tau_{n1}^2\}$ and group of member tasks $\tau^1 = \{\tau^1_{n1+1} \ldots\ldots, \tau^1_n\}$ to be scheduled on a processor Ψ with λ homogeneous cores.

Output: Real-time MC schedule

1: $\tau(\Psi_j) \leftarrow 0$, for all j = 1,..., λ.
2: for i ← 1 to n_1 ►*for executive tasks*
   a) Calculate the utilization $U_i^2(\tau)$
   b) Order task set in descending order of $U_i^2(\tau)$
   c) Order tasks with same $U_i^2(\tau)$ value in descending order of $U_i^1(\tau)$
3: for i ← n_1 to n ►*for member tasks*
   a) Calculate the utilization $U_i^1(\tau)$
   b) Order tasks in descending order of $U_i^1(\tau)$
   c) Order tasks with same $U_i^1(\tau)$ value in ascending order of its inter-arrival time (P_i)
4: while τ^2 != 0 and τ^1 != 0 do ► *Clustering*
5: $\tau_s \leftarrow$NIL
6: if τ^1 != 0 then
7: $\tau_s \leftarrow$ The first member task from τ^1
8: if τ^2 !=0 then
9: $\tau_s^* \leftarrow$The first executive task from τ^2
10: if τ_s = NIL ||
τ_s != NIL && $U^1(\tau_s) < U^2(\tau_s^*)$ then
11: $\tau_s \leftarrow \tau_s^*$
12: Remove τ_s from its group ► *Bin-packing*
13: Select a core $\Psi_j \in \Psi$ that has a minimum cluster utilization for $\tau(\Psi_j) \cup \{\tau_s\}$
14: if the cluster utilization for tasks
$\tau(\Psi_j) \cup \{\tau_s\} \leq 1$ then
15: $\tau(\Psi_j) \leftarrow \tau(\Psi_j) \cup \{\tau_s\}$
16: else
17: return FAILURE
18: return SUCCESS

8. Experiments and Results

We investigate and empirically assess the schedulability of C-PEDF by conducting extensive simulation experiments. To evaluate the effectiveness of our suggested scheme, we experimentally compare the performance of C-PEDF against the following partitioned MC scheduling approaches:

- DC-RMS: a scheduling algorithm from [20] in which the workloads are ordered in descending order based on their level of importance and higher priorities are allocated to workloads with lower inter-arrival time (i.e., Decreasing Criticality-Rate Monotonic Scheduling);
- DU-RMS: a scheduling algorithm from [20] in which the workloads are arranged in descending order based on their utilization and higher priorities are allocated to workloads with lower inter-arrival time (i.e., Decreasing Utilization-Rate Monotonic Scheduling);
- DC-AOPA (Decreasing Criticality-Audsley's Optimal Priority Assignment): an approach from [20] in which priorities are allocated in order, from lowest to highest. In each cycle, the algorithm finds an appropriate workload for the next priority level. If so, it gives that priority, then allocates the subsequent priorities to the other workloads in a similar way;
- DU-AOPA (Decreasing Utilization – AOPA): an approach from [20] in which the tasks are scheduled according to their utilization and AOPA algorithm is used for priority allocation;
- MC-PARTITION: EDF-based approach relies on Virtual Deadlines from [29];
- EY-FF: an enhanced Ekberg and Yi (EY) algorithm [13] exploiting First-Fit (FF) packing policy from [15];
- MPVD (Mixed-criticality Partitioning with Virtual Deadlines): a further extension of EY algorithm with the hybrid packing model from [15];
- E-MPVD (Enhanced-MPVD): MPVD enhanced by the heavy low-criticality task cognizant assignment strategy in [15]; and
- MPVD-OPT: E-MPVD further enhanced by the optimized virtual deadline tuning from [15].

We evaluate our proposed algorithm with the following objectives: (1) to evaluate the effectiveness of C-PEDF in terms of acceptance ratio (i.e. the fraction of the number of tasks that are deemed to be MC-schedulable by the algorithm to the total number of the tasks in the experiment); (2) to analyse how well C-PEDF can defend low-level tasks when an executive task exhibits its critical behaviour; (3) to examine the impact of probability of the generated workload exhibit high-level execution behaviour on Deadline Miss Ratios (DMR) of member tasks; and (4) to analyse the impact of overprovisioning in terms of schedulability on a single-core processor by enabling a member task to present in more than one clusters. The impact factor of member tasks is considered as a metric to evaluate the C-PEDF performance. We specify the impact factor of member tasks as the ratio of such workloads for which at least one job is discarded or violates its timing constraint since an executive task exhibits its critical behaviour. The ratio of low-level tasks that violate their timing constraints is called as DMR.

8.1. Workload

For our experiments, we implement a UUniFast random task generator as used in [14]. Each data-point in the curves was derived by at least 2000 random tasks. Here, the parameter, Prob(τ_j^2) represents the probability of a task to be an executive task. Initially, we consider a system with an equal amount of high- and low-level workloads and set Prob(τ_j^2) = 50%. The period P_i was selected within the range of [1, 500] time units. The low-level WCET C_i^1 of τ_i was derived from [0.05 × P_i, 0.5 × P_i] and, if τ_i was an executive task, its high-level WCET C_i^2 was derived from [2 × C_i^1, 4 × C_i^1]. The average utilization $U_{av}(\tau)$ is calculated as:

$$U_{av}(\tau)=\frac{U_i^1(\tau)+U_i^2(\tau)}{2\times\lambda} \qquad (20)$$

where $U_i^1(\tau)$ and $U_i^2(\tau)$, are the sum of utilizations of low- and high-level workloads correspondingly. Table 3 exemplifies the estimation of these cumulative utilization parameters for a different criticality level of tasks. Each task set is created with a normalized average utilization $U_{av}^*(\tau)$ with a tolerable range of errors. Based on the scheduling parameters, workloads are

generated uniformly until the following constraints on utilization bound were met:

(i) $U^*_{av}(\tau) - 0.005\lambda \le U_{av}(\tau) \le U^*_{av}(\tau) + 0.005\lambda$
(ii) $U_i^1(\tau) \le U_\lambda$; and
(iii) $U_i^2(\tau) \le U_\lambda$

where $U^*_{av}(\tau) \in \{0.5, 0.55, 0.6, \ldots, 0.95\}$ and $\lambda \in \{2,4,8\}$. Since the estimation of DMR and the impact factor involve expensive simulations, we select only 50 tasks arbitrarily with $U^*_{av}(\tau) \in \{0.65, 0.75, 0.85, 0.95\}$. We execute each simulation for 10000 msec. In order to assess the impact of overprovisioning on the schedulability under varying number of clusters that a member task can fit into, we select the value of Prob(τ_j^2) to 90%, and $U_i^1(\tau)$ within the range of [0.01, 0.2] to produce more executive tasks.

8.2. MC schedulability

In Fig.9a - 9c, we depict the fraction of task sets successfully scheduled as a function of the normalized average utilization (load) of the various heuristics in 2-core, 4-core, and 8-core processors correspondingly. We first observe that as the load increases, the acceptance ratio decreases for all compared schemes. This is due to the fact that as the average utilization rises, there are more tasks in the platform that needs to be serviced. Therefore, there is a higher load on the processor, and as a result, more workloads violate their timing constraints.

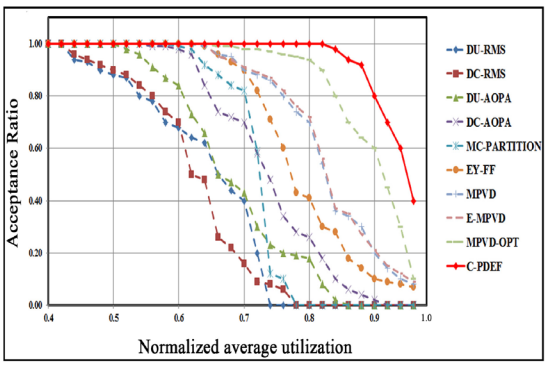

Fig.9(a). MC schedulability for 2-Core systems

Consequently, we observe that schemes using AOPA provide better performance as compared to schemes using RMS policies. Therefore, these results prove that the choice of the priority allocation method plays a vital role to enhance schedulability. The results for 4- and 8-core processor reveal that the success ratio of EY-FF reduces as the core count increases. This is due to the uneven assignment of scheduling parameters in EY decreases the possibility of optimizing virtual deadline.

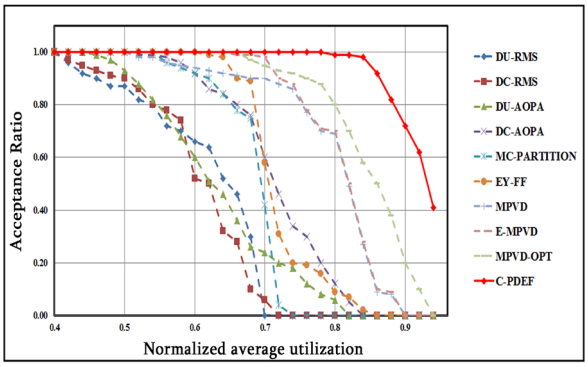

Fig.9(b). MC schedulability for 4-Core systems

The graphs in Fig. 9c show the acceptance ratio against the system load under different scheduling algorithms on an 8-core processor. We observe that the algorithms MPVD, E-MPVD, and MPVD-OPT provide better performance as compared to EY-FF by distributing the workloads between cores. E-MPVD outperforms MPVD by allocating substantial low-level workloads to cores before high-level workloads, but the success ratio of these two approaches follow the same trend when the system comprises limited low-level tasks.

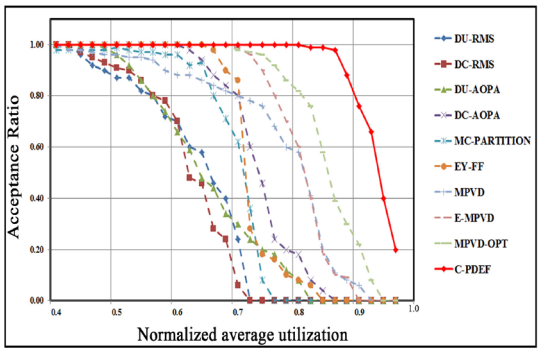

Fig.9(c). MC schedulability for 8-Core systems

MPVD-OPT targets to increase the acceptance ratio by means of a virtual deadline optimizing technique, and its acceptance ratio is greater than MPVD and E-MPVD; however, these optimizing technique does not consider the low-level workloads, so it jeopardizes the acceptance ratio of the system. Therefore, the

performance of MPVD-OPT is inferior to C-PEDF for all 2-core, 4-core and 8-core processor with higher average utilization. From the results, we can observe that many more tasks can be scheduled under C-PEDF with a given number of cores. The proposed algorithm constantly exhibits a substantial enhancement in the acceptance ratio and outperformed all state-of-art algorithms. Obviously, there is a wide performance gap among the schedulability of C-PEDF and those of other approaches. This is because, with an increase in the number of workloads, there are more opportunities for our C-PEDF to select suitable clusters for each member task.

8.3. Effect of H-Mode behavior on member tasks

In our study so far, we have fixed prob(τ_j^2) = 50%. Next, we evaluate the impact factor of the member tasks for varying fractions of executive tasks to get into H-Mode. In Fig. 10, results are plotted for the fixed $U_{av}^{*}(\tau)$ ∈ {0.65, 0.75, 0.85, 0.95} on a 4-core platform. We observe that only a certain ratio of member tasks is influenced and that this ratio increases with the ratio of executive tasks that show H-Mode behaviour. The reason is, with C-PEDF, the H-Mode behaviour of an executive task has an impact only on member tasks in the same cluster, but not in other clusters.

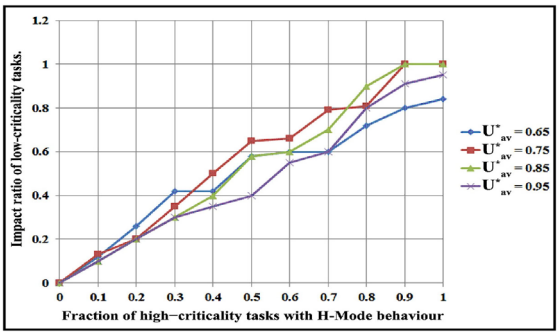

Fig.10. Impact factor of member tasks

We observe some important properties: (i) The relationship is not exactly linear since a group of low-level workloads are clustered with multiple executive tasks, and can thus be influenced by any one of these clusters. (ii) Once all the admitted workloads show H-Mode behaviour (i.e., prob(τ_j^2) = 100%), there are still certain member tasks in the system that are not influenced. This is because the cores having only member tasks cannot be affected by the H-Mode behaviour of other clusters. (iii) As expected, the impact ratios are not affected by varying average utilization. This is due to the proportion of member tasks that are influenced hinges mainly on the clustering process and Prob(τ_j^2).

8.4. Real-time performance of member tasks

The impact of H-Mode behaviour on the DMR of member tasks in a 4-core processor is plotted in Fig. 11. As expected, there is no task misses its deadline when prob(τ_j^2) = 0: when the system is in L-Mode, the C-PEDF can guarantee that the system is schedulable. As the prob(τ_j^2) increases, the DMR of the member task is also increased slowly; this is because that the H-Mode execution behaviour of a task can only influence other members within the cluster, but not tasks in other clusters. Furthermore, since the isolation among clusters allows the system to transit back to L-Mode when all executive tasks show L-Mode behaviour, the fraction of a deadline miss of the member tasks can be preserved small even for long-running applications.

It is evident that the DMR plots corresponding to various average processing capacities cross each other, that is, a cluster with a greater value of $U_{av}^{*}(\tau)$ will have a smaller DMR than that of a cluster with a lesser value of $U_{av}^{*}(\tau)$. This is anticipated because the DMR can become lesser as we generate more member tasks, which can be the case when $U_{av}^{*}(\tau)$ rises. It is important to note that the other scheduling approaches reject the entire low-level workloads immediately an executive task shows H-Mode behaviour, so they provide no service assurance for member tasks in the H-Mode.

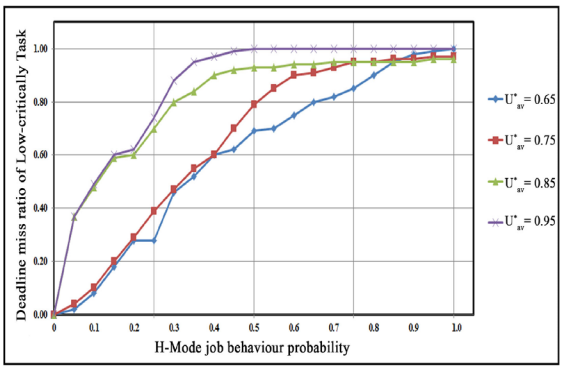

Fig.11. Deadline miss ratio of member tasks.

8.5. *Impact of resource overprovisioning on schedulability*

We continue to present the impact of resource overprovisioning on acceptance ratio. Fig. 12 shows the impact of allowing member tasks to exploit the overprovisioning of various executive tasks on the schedulability of the given task on a single-core system. Here, $N^{m\_cluster}$ is the upper bound on the total number of clusters available for each member task.

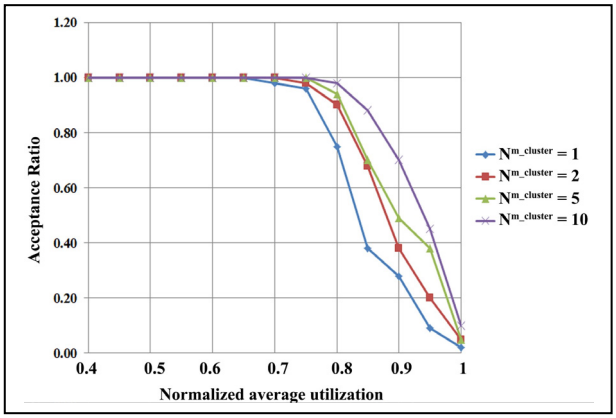

Fig.12. Effect of overprovisioning on schedulability

As shown in Fig. 12, the acceptance ratio increases with $N^{m\_cluster}$. Furthermore, the success ratio gap tends to large as the load increases, which further specifies the usefulness of exploiting overprovisioning of executive tasks.

9. Future work

The additional research effort is required before our scheduling approach could be considered appropriate for real tasks. There are two potential directions of our future work: application studies and architecture improvements.

Application studies: We plan to apply our scheduling approach in LITMUS^{RT} (LInux Testbed for MUltiprocessor Scheduling in Real-Time systems) on an Intel® Xeon® Processor E7440 platform [30]. It is a 32-bit processor consists of 4 cores on a single chip operating at 2.40 GHz, with 16 MB L2 cache per processor and 4 GB of RAM. Each core can process 4 logical threads in parallel. The implementation of extensive measurements campaigns using this testbed will be intended for performance assessments that will give more constructive feedback on the pragmatism of our approach.

Architecture improvements: A substantial amount of improvements to our approach would increase the range of real-time tasks that can be supported on multicore systems. A leading example of this is efficient resource sharing across applications of different safety criticalities. An additional active field of interest is facilitating adaptivity, which is deemed to be an essential concern in future industrial safety-critical systems, which themselves must adapt different run-time operating conditions (environments) dynamically. For instance, in an unmanned aircraft, when previously-undetectable opponent radar stations are found, it might be beneficial to give rise to the share of computing resources of a pathfinding and route planning processes. On the other hand, if an unfriendly guided missile were to be spotted, in order for appropriate action or communication to be performed more quickly, the facility to enable a mode-switch (wherein a new set of tasks replaces those currently being scheduled) might be required. We would like to point out that even though our proposed approach considers a dual criticality system, it is likely to extend this approach to more than two levels. We intend to extend our algorithm to investigate the run-time overhead limitations of the proposed algorithm to obtain a global view of the gains of our approach. Finally, we plan to extend our scheduling strategy to execute multi-criticality tasks on a heterogeneous platform in a hierarchical manner.

10. Conclusions

We have proposed a Clustering-based partitioned EDF scheduling algorithm for scheduling dual-criticality tasks on a multicore platform. In our proposed C-PEDF algorithm, each high-level workload is coalesced in a cluster (to facilitate separation) along with a group of less critical tasks (to maximize schedulability of tasks while preserving the criticality assurance). Within each cluster, tasks are scheduled under Enhanced dual-mode scheduling policy to improve the service level of high-level tasks without jeopardizing the schedulability of low-level tasks. Clusters are scheduled under Earliest Deadline First (EDF) scheduling approach.

Our approach enforces strong temporal isolation between high-criticality tasks to alleviate all inter-task

interferences, and it allows more lower-criticality workloads to satisfy their timing requirements. This is due to (i) the undesirable service intervention of high-level workloads on low-level workloads is considerably decreased; and (ii) low-level workloads are not continuously dropped, but rather received sufficient execution when the high-level workload in the same cluster exhibits its critical behaviour. We conduct a schedulability test for the proposed technique, and we demonstrate how workloads can be clustered by means of evolutionary intelligence technique, namely Mixed Integer Nonlinear Programming (MINLP) model. Extensive simulation results reveal that our algorithm significantly outperforms other approaches both in acceptance ratio and the impact factor of low-level tasks.

Reference

1. K. P. Valavanis, *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy*, (Springer Publishing Company, Incorporated, 2007, vol.33).
2. European Organisation for Civil Aviation Equipment. (1992). DO-178B, *Software Consideration in Airborne Systems and Equipment Certification*, EUROCAE.
3. N. Guan, P. Ekberg, M. Stigge and W. Yi., Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems, in *Proc. 32nd Real-Time Systems Symposium*, IEEE, (2011) pp. 13–23.
4. B. B. Brandenburg, J. M. Calandrino and J. H. Anderson, On the scalability of real-time scheduling algorithms on multicore platforms: A case study, in *Proc. Real-Time Systems Symposium*, IEEE, (2008) pp.157–169.
5. U. C. Devi and J. H. Anderson, Tardiness bounds under global EDF scheduling on a multiprocessor, *The Journal of Real-Time Systems*, 38(2) (2008)133–189.
6. H. J. Leontyev and H. Anderson, Generalized tardiness bounds for global multiprocessor scheduling, *The Journal of Real-Time Systems*, 44(1) (2010) 26–71.
7. A. Mills and J. H. Anderson, A stochastic framework for multiprocessor soft real-time scheduling, in *Proc.16th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE, (2010), pp. 311–320.
8. S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in *Proc. 28th IEEE International Real-Time Systems Symposium*, IEEE, (2007), pp. 239–243.
9. F. Dorin, P. Richard, M. Richard and J. Goossens, Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities, *Real-Time Systems,* 46(3) (2010) 305–331.
10. S. Baruah, H. Li and L. Stougie, Towards the Design of Certifiable Mixed-criticality Systems, in *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE, (2010), pp. 13–22.
11. S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow and L. Stougie, Scheduling real-time mixed-criticality jobs, in *IEEE Trans. Computers,* 61(8) (2012) 1140–1152.
12. H. Li and S. Baruah, An algorithm for scheduling certifiable mixed-criticality sporadic task systems, in *Proc. 34th Real-Time Systems Symposium*, IEEE, (2010), pp. 183–192.
13. P. Ekberg and W. Yi, Bounding and shaping the demand of mixed-criticality sporadic tasks, in *Proc. Euromicro Conference on Real-Time Systems,* (2012) pp. 135–144.
14. H. Li and S. Baruah, Global mixed-criticality scheduling on multiprocessors, in *Proc. 24th Euromicro Conference on Real-Time Systems,* (2012), pp. 166–175.
15. C. Gu, N. Guan, Q. Deng and W. Yi, Partitioned mixed-criticality scheduling on multiprocessor platforms, in *Proc. Design, Automation and Test in Europe Conference and Exhibition,* IEEE, (2014), pp. 1– 6.
16. P. Ekberg and W. Yi, Bounding and shaping the demand of generalized mixed-criticality sporadic task systems, *Real-time systems*, 5(1) (2014) 48–86.
17. M. Mollison, J. Erickson, J. Anderson, S. Baruah and J. Scoredos, Mixed-criticality real-time scheduling for multicore systems, in *Proc. 10th IEEE Int. Conf. Computer and Information Technology,* (2010), pp. 1864–1871.
18. 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 *Proc. 7th ACM Int. Conf. Embedded Software,* (2009), pp. 235–244.
19. J. H. Anderson, S. K Baruah and B. B. Brandenburg, Multicore operating-system support for mixed criticality, in *Proc. Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, April 2009.
20. O. Kelly, H. Aydin and B. Zhao, On partitioned scheduling of fixed-priority mixed-criticality task sets, in *Proc. 10th Int. Conf. Trust, Security and Privacy in Computing and Communications,* (2011), pp.1051–1059.
21. A. Burns, System mode changes-general and criticality-based, in *Proc. of the 2nd Workshop on Mixed Criticality Systems,* (2014), pp. 3–8.
22. S. Petters, M. Lawitzky, R. Heffernan and K. Elphinstone, Towards real multi-criticality scheduling, in *Proc. 15th IEEE Int. Conf. Embedded and Real-Time Computing Systems and Applications,* (2009), pp. 155–164.
23. S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van der Ster and L. Stougie, The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems, in *Proc. Euromicro Conference on Real-Time Systems,* (2012), pp. 145–154.
24. M. L. Dertouzos and A. K.Mok, Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Engineering*, 15(12) (1989) 1497–1505.

25. A. Mahajan and T. Munson, *Exploiting second-order cone structure for global optimization*, Technical Report ANL/MCS-P1801-1010, Argonne National Laboratory, 2010.
26. A. Mahajan, S. Leyffer and C. Kirches, *Solving mixed-integer nonlinear programs by QP-diving*. Preprint ANL/MCS-P2071–0312, Argonne National Laboratory, 2012.
27. R. Fourer, D. M. Gay and B. W Kernighan, *AMPL: A Modelling Language for Mathematical Programming*. Duxbury Press, Brooks/Cole Publishing Company, 1993.
28. R. Fletcher and S. Leyffer, Solving mixed integer nonlinear programs by outer approximation, *Mathematical Programming*, 66(1) (1994) 327–349.
29. S. Baruah, B. Chattopadhyay, H. Li and I. Shin, Mixed-criticality scheduling on multiprocessors, *Real-Time Systems*, 50(1) (2014) 142–177.
30. J. Calandrino, H. Leontyev, A. Block, U. Devi and J. Anderson, LITMUS^{RT}: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers, in *Proc. 27th IEEE Real-Time Systems Symposium,* (2006) pp. 111–123.