Online Admission of Non-Preemptive Aperiodic Mixed-Critical Tasks in Hierarchic Schedules

Ali Syed, Gerhard Fohler
Technische Universität Kaiserslautern, Germany
{syed, fohler}@eit.uni-kl.de

Daniel Gracia Pérez
Thales Research & Technology, France
daniel.gracia-perez@thalesgroup.com

***Abstract*—Operational safety is one of the major issues in modern industrial applications. For safety critical systems, the verification and validation efforts can be reduced using temporal isolation enforced by hierarchic schedules. In 2012, Lackorzyński et al. demonstrated that reserving bandwidth for event-triggered (ET) activities in hierarchic schedules may lead to significant bandwidth loss. In contrast to the bandwidth reservation techniques, online admission of ET activities can help reduce bandwidth losses. However, the state-of-the-art approaches for online admission of ET activities (i) cannot be utilized in hierarchic systems and (ii) incur significant scheduling overheads.**

In this paper, we present a flexible scheduler design for online admission of non-preemptive aperiodic tasks in a hierarchic time-triggered environment. Our approach circumvents the bandwidth loss issue in hierarchic systems, and can be implemented on top of a variety of hypervisor interfaces. In order to provide offline guarantees, we also provide a necessary test for our approach. Through evaluation, we demonstrate that our approach efficiently utilizes processor bandwidth and only incurs small overheads for a safety critical system. The experimental study also shows that our approach is comparable, in terms of execution overheads, to the state-of-the-art preemptive non-hierarchic approaches.

I. Introduction

Modern industrial applications are subject to several safety constraints. Verification and validation (V&V) processes utilized for such applications define if the product can be deployed for the desired application. However, the V&V process requires rigorous evaluation of the system, and therefore, it increases in complexity as the system size increases. Partitioning or virtualization enables certification of a sub-system irrespective of the behaviour of other sub-systems, and as a result, reduces the V&V efforts [1].

In addition to the safety constraints, a computing system may need to handle several real-time constraints depending on the application requirements. The characteristics of such systems strongly depend on the used model of computation, i.e. time-triggered (TT) or event-triggered (ET). In real-time systems, periodic tasks can easily be serviced using TT mechanisms due to predictable repetitive arrival patterns. However in the case of aperiodic tasks, TT mechanisms like offline bandwidth reservation may lead to over-provisioning, and therefore, may increase cost per feature. The over-provisioning problem becomes increasingly prohibitive with virtualization technologies as these technologies may lead to significant bandwidth loss [2]. Therefore, new algorithms need to be developed which provide efficient integration of periodic and aperiodic activities in hierarchic systems.

978-1-5386-1898-1/17/$31.00

Besides bandwidth reservation, a number of techniques are proposed over the last few decades to integrate TT and ET activities. For instance, Slack Stealing [3] is used for online admission of aperiodic tasks in fixed-priority systems, whereas, Slot-Shifting [4] is used with EDF (Earliest Deadline First). The basic operating principle for both of these approaches is to use the information about the amount and distribution of free resources to accommodate aperiodic tasks. A few years after the proposal of the Slot-Shifting algorithm, its extensions were designed to compensate for sporadic tasks [5], non-preemptive aperiodic tasks in a preemptive scheduling environment [6] and mixed-criticality tasks [7]. Although these extensions are viable solutions, they need to frequently keep track of free resources, they are computationally expensive and are only applicable on TT systems with sparse time-base [8]. Moreover, none of these extensions can be applied on hierarchic scheduling schemes.

In this paper, we present an algorithm which defines availability of free resources in an offline TT scheduling table of non-preemptive tasks, and employs the availability information online to service non-preemptive aperiodic tasks in a partitioned system; we call this algorithm 'Job-Shifting'. Although non-preemptive scheduling offers several benefits (e.g. reduced I/O delays [9], precision in the estimation of WCET [10], etc.), it is an NP-Hard problem [11]. Consequently, the non-preemptive versions of EDF (e.g. npEDF) are not optimal [9]. Therefore, the job-shifting algorithm is not designed to confine the application integrator to the EDF strategy, unlike its predecessors [4], [7], [6], and can utilize either online or offline scheduling strategies. Furthermore, job-shifting can be used with sparse and dense time-bases [8] and does not require computationally expensive slot-based record keeping mechanism.

Contributions: In this paper, we present the job-shifting algorithm which can be used inside a partition to provide online aperiodic admission in a non-preemptive execution environment. We highlight the advantages and disadvantages of the job-shifting algorithm and provide a necessary condition for schedulability. Through simulation, we demonstrate that the proposed algorithm efficiently utilizes free resources in hierarchic scheduling systems. Moreover, we evaluate the overheads introduced by the job-shifting algorithm on Zynq

ZC706 board [12], and demonstrate that the incurred overheads (i) are small and (ii) are comparable to the overheads incurred by the state-of-the-art *preemptive* approaches.

The reminder of the paper is organized as follows; Sec. II provides a summary of the related works. Sec. III introduces the system model. Sec. IV defines the job-shifting algorithm, and highlights the viability of the algorithm. Sec. V provides the experimental description utilized for job-shifting efficiency evaluation. Sec. VI presents the overhead evaluation experiment with description of the safety critical demonstrator and the job-shifting implementation in the EU FP7 DREAMS project [13]. Finally, the paper is concluded in Sec. VII.

II. Related Work

Lehoczky et al. defined slack-stealing algorithm to service soft [3] and hard [14] aperiodic task in fully-preemptive fixed-priority systems. In this algorithm, they precomputed and stored the *slack function* of the task-set and used it online to service aperiodic tasks. The slack-stealing algorithm was later extended by Tia et al. [15] to efficiently compute the *slack function* online, since the size of the slack function table may be too large, depending on the periods of the tasks.

Similarly, Fohler [4] defined slot-shifting algorithm for aperiodic admission in fully-preemptive TT dynamic-priority systems (specifically EDF) with sparse time-base [8]. In this algorithm, *capacity (or slack) intervals* and their *spare capacities* are calculated offline and stored in a table to be used online to service firm and soft aperiodic and sporadic tasks [5]. The slot-shifting algorithm requires a significant amount of memory and utilizes an online mechanism to keep track of used resources activated at each *slot* boundary [8].

Recently, Schorr [6] extended the original slot-shifting algorithm to allow admission of non-preemptive aperiodic tasks in a preemptive schedule. The extension utilized the original precomputed offline scheduling table, however the aperiodic admission test (termed *acceptance test* [4]) for non-preemptive aperiodic tasks was modified to calculate enough consecutive slots to execute the released aperiodic job. Moreover, the *guarantee algorithm* [4] was modified to improve the response-time at the cost of flexibility [6]. Similar to the original slot-shifting algorithm, the extension by Schorr requires a significant amount of memory and uses an online slot based record keeping mechanism. Moreover, the slot-shifting extension proposed by Schorr has higher complexity and larger run-time overheads.

Similarly, Theis [7] extended the slot-shifting algorithm to support mixed-critical tasks (based on Vestal's MCS model [16]) and mode changes. Similar to the original slot-shifting algorithm, the extension by Theis is based on EDF, however the memory requirements (and therefore the overheads) are increased proportional to the number of modes or criticalities.

In this paper, we present job-shifting algorithm, which can be used in industrial hierarchical mixed-critical systems (*not* based on Vestal's MCS model [16]) to admit non-preemptive aperiodic tasks. Unlike its predecessors [4], [6], [7], the job-shifting algorithm does *not* require (i) slot-based record keeping mechanism, (ii) sparse time-base [8] or (iii) EDF scheduling. The job-shifting algorithm respects separation of concerns (through partitioning), incurs small overheads compared to its predecessors and provides better response-times for aperiodic activities. However, job-shifting only supports non-preemptive scheduling, which is NP-Hard even for the simple case of independent tasks with implicit deadlines [11].

III. System Model

A partitioning kernel or hypervisor is used to provide strict temporal and spatial isolation ([17], [18], [19]), which enables independence of safety functions between applications. In order to fulfill strict safety requirements, cyclic-executive scheduling [20] is assumed to be the inter-partition scheduling strategy. We assume a uni-processor non-preemptive execution environment. Moreover, all activities in the system are assumed to be triggered by the passage of time, i.e. Time-Triggered with sparse or dense time-base [8]. For digital computing systems, the sparse time-base is implemented by the scheduler using a scheduling quantum significantly larger than the processor clock cycle length, e.g., a slot in slot-shifting [4]. However, when the scheduler does not enforce a sparse time-base, the system is subjected to dense time-base (aka 'as fast as possible') where the scheduling quantum is equal to the processor clock cycle length.

In order to apply job-shifting to a partition p_i in a partition set P, it is assumed that the intra-partition scheduling employs TT scheduling tables with a simple online dispatcher (relaxed in Sec. IV-G4), and contains both periodic and aperiodic tasks. The idle time inside a partition p_i is also assumed to be non-zero (discussed further in Sec. IV-F). The time when the partition p_i is not available to service applications is defined by the blocking set V_i. Each blocking $v \in V_i$ is defined by the tuple $\langle b, m \rangle$, where b and m represent the absolute beginning and termination time of blocking v, respectively.

A task-set Γ is defined as a collection of tasks. Each task $\tau \in \Gamma$ is defined by the tuple $\langle \phi, C, T, D, Y \rangle$, where ϕ represents the task phase, and C represents the worst-case execution time (WCET) of the task τ. When the task is periodic, the parameter T defines its period; otherwise, $T = \infty$. Moreover, D defines the relative constrained deadline (i.e. $D \leq T$), and Y represents the task criticality (i.e. the safety level, e.g. DAL-B). An aperiodic task release is usually detected by checking a peripheral specific (interrupt) register, e.g. a button is pressed, a message is received. It is necessary to mention here that in TT systems, only timer interrupts are enabled. A task $\tau \in \Gamma$ consists of infinite jobs j, each of which is defined by the tuple $\langle r, d \rangle$, where r represents the absolute job release time, and d defines the absolute job deadline. The jobs j are assumed non-preemptive, however they can be paused at the partition boundary (further in Sec. VI-A3).

The scheduling table S_p for a partition p is constructed prior to the job-shifting offline phase (discussed in Sec. IV-B2) for

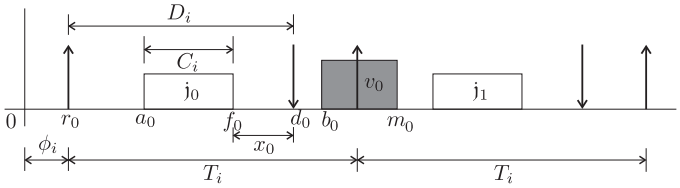


Fig. 1: Model parameters for a task τ_i with blocking v_0

Notation	Meaning	Notation	Meaning
ϕ	Task phase	a	Job activation (absolute)
T	Task period	f	Job finish (absolute)
Y	Task criticality	r	Job release (absolute)
C	Task WCET	d	Job deadline (absolute)
x	Job flexibility	D	Task deadline (relative)
SC	Scheduling cycle	R	Aperiodic room

TABLE I: Summary of the used notations

the length of the scheduling cycle (SC). The length of SC is defined by the following equation;

$$\text{SC} = \begin{cases} [0, \text{LCM}] & \forall \tau : \phi = 0 \\ [0, \phi_{max} + 2 \times \text{LCM}] & otherwise \ [21] \end{cases} \quad (1)$$

where, ϕ_{max} represents the maximum phase ϕ of all tasks $\tau \in \Gamma$, while LCM represents the least common multiple of all the periods T of periodic tasks. For each job in SC, the scheduling table S_p defines the absolute job activation time a and the absolute job finish time f. It is assumed that the partition scheduling table S_p, available as input to job-shifting algorithm, is a valid feasible schedule. The tasks $\tau \in \Gamma$ may have precedence and/or mutual exclusion constraints. However, these constraints are assumed to be resolved either by modifying the job release time r and deadline d or by constructing the scheduling table S_p. An example scheduling table for a periodic task τ_i with blocking v_0 is shown in Fig. 1, while a summary of the used notations is provided in Table I.

IV. Job-Shifting Algorithm

In this section, we provide details of the job-shifting algorithm to admit non-preemptive aperiodic tasks in flat and hierarchical scheduling models. Without loss of generality, in this section we assume that the sparse time-base [8] is used. Moreover, all the jobs are assumed to execute for the complete worst-case execution time C every time (relaxed in Sec. IV-G3).

A. Methodology

In order to apply the job-shifting algorithm, a feasible offline scheduling table S_p for partition p is required. To construct S_p, the tasks $\tau \in \Gamma$ are allocated to the processing nodes and the partitions P. The periodic tasks are then unrolled to create jobs for the length of SC and the scheduling table S_p is constructed utilizing the desired scheduler such that the feasibility is ensured. For the offline scheduled partitions, this necessarily means that the values for a_i and f_i are defined for each job j_i inside the partition such that $\forall \mathrm{j}_i \in S_p : r_i \leq a_i < d_i \wedge a_i < f_i \leq d_i \wedge f_i - a_i \geq C_i$.

Once the offline scheduling table S_p for a partition is ready, the job-shifting algorithm can be applied, which is divided in two phases; an offline phase and an online phase. In the offline phase, a parameter, we call the flexibility coefficient x_i, for each job $\mathrm{j}_i \in S_p$ is calculated (for example, see x_0 in Fig. 1). We define the flexibility coefficient x_i as:

Definition IV.1. The flexibility coefficient x_i for job $\mathrm{j}_i \in S_p$ defines *the maximum delay, which can be added to the absolute activation time a_i of job j_i without changing the execution order of jobs in S_p and without missing any deadline.*

During the online phase of job-shifting algorithm, the scheduler checks the arrival of the aperiodic job(s). When a new aperiodic job is detected, the guarantee test is performed. If the guarantee test succeeds, the new aperiodic job is adjusted in the partition schedule by invoking the guarantee algorithm. However, if the guarantee test fails, the aperiodic job is added to the best-effort queue. Upon each activation of the scheduler, a job on the best-effort queue can be executed if there exists enough aperiodic room R_i, which we define as follows:

Definition IV.2. The aperiodic room R_i defines *the maximum contiguous processing node time, which can be spared for executing aperiodic job(s) prior to the activation of job j_i, without missing any deadline in the system.*

On account of the above definition, it can be observed that the best-effort queue is not a background queue. Instead, a job on the best-effort queue is executed as soon as enough aperiodic room R_i can be reserved (further in Sec. IV-F).

B. Flat Scheduling Model

In this section, we assume that there exists only one partition which is available to service tasks at all times, i.e. $V_p = \emptyset$. Furthermore, the finish time f_i for a job j_i is not defined by the scheduling table S_p, instead f_i is directly calculated by the equation $f_i = a_i + C_i$.

1) Offline Phase: As mentioned earlier, during the offline phase, the flexibility coefficient x_i for each job $\mathrm{j}_i \in S_p$ is calculated starting from the job j_s with the maximum activation time a_s and ending at the job j_e with the minimum activation time a_e. Assuming $a_{s+1} = d_s$ and $x_{s+1} = 0$, for each job j_i the following steps are performed;

i) Calculate the parameter O_i, which defines the overlap between the flexibility windows $[a, d]$ of job j_i and job j_{i+1}, using the following equation;
$$O_i = max(0, d_i - a_{i+1}) \quad (2)$$
ii) Calculate the flexibility coefficient x_i using the following equation;
$$x_i = d_i - a_i - C_i - O_i + min(x_{i+1}, O_i) \quad (3)$$

2) Online Phase: After finishing the offline phase, all the parameters for each job are passed to the online scheduler, which is activated at each job activation time a, finish time f and (when idle) at the release of a new aperiodic job.

The guarantee test is performed when the scheduler gets activated at time t and an aperiodic job release is detected. For the guarantee test, the job j_n is defined as the next job

to be activated from the scheduling table S_p, while the job j_l is defined as the first job activated after the deadline of the released aperiodic job j_{ap}. During the guarantee test, the aperiodic room R_i for each job i from j_n to j_l is calculated using the following set of equations;

$$\begin{aligned} s_i &= \begin{cases} a_{i-1} + C_{i-1}, & n < i < l \\ t, & i = n \end{cases} \\ R_i &= a_i - s_i + min(d_{ap} - a_i, x_i) \end{aligned} \tag{4}$$

where, s_i represents the start of the idle time before job j_i. The guarantee test passes when, for any job $j_i | n \leq i < l$, the aperiodic room R_i is larger than or equal to C_{ap}. If the guarantee test passes, the aperiodic job j_{ap} can be guaranteed to finish before its deadline d_{ap} without missing any other deadline in the system. Once the guarantee test passes, the guarantee algorithm is activated. The guarantee algorithm requires the following steps to be performed:

i) Insert the released aperiodic job j_{ap} in the schedule S_p before job j_i , i.e. $a_i = a_{ap} = s_{i+1}$.
ii) For each job j_k, such that $k > i \wedge a'_k > a_k$, perform the following steps starting from j_{i+1};
   a) $a'_k = a_k + max(a_{k-1} + C_{k-1} - a_k, 0)$
   b) $x_k = x_k - (a'_k - a_k)$
   c) $a_k = a'_k$
iii) For each job j_p, such that $p \leq i$, calculate x_p similar to the offline phase starting from j_i and ending at j_n. The process of modifying x_p stops when the old x_p is equal to the new x_p.

At the scheduler activation time t, the scheduling decision can be made after handling the released aperiodic job(s). If the best-effort queue is empty, the next job with $a = t$ is scheduled from S_p. When there exists no such job, the processor is left idle. If there exists a job j_b in the best-effort queue and the next job aperiodic room $R_n \geq C_b$, the guarantee algorithm is performed for job j_b with $a_b = t$ and it is scheduled.

C. Hierarchic Scheduling Model

In this section, we assume that there exist multiple partitions in the system and the job-shifting algorithm is enabled in partition $p \in P$, i.e. $V_p \neq \emptyset$. For hierarchic scheduling model, we need to define two operators; the Blocking operator $B(p, q)$ and the Adjust operator $A(u)$. The blocking operator $B(p, q)$ returns the sum of the partition blocking duration between the interval (p,q]. The operator $A(u)$ modifies the time instant u such that it does not lie within the partition blocking time V. When the time instant u denotes the job finishing time f_i, the operator $A(u)$ also makes sure that the duration between the job activation time a_i and the job finish time f_i is enough to complete the job, i.e. $f_i = a_i + B(a_i, f_i) + C_i$. Unlike the flat scheduling model, the finish time f_i for a job j_i in the hierarchic scheduling model is also defined by the scheduling table S_p. Moreover, it is assumed that the parameters a_i and f_i are adjusted offline using the adjust operator $A(u)$.

1) Offline Phase: Similar to the offline phase of flat scheduling model, the flexibility coefficient x_i for each job $j_i \in S_p$ is calculated during the offline phase of hierarchic scheduling model starting from the job j_s with the maximum activation time a_s and ending at the job j_e with the minimum activation time a_e. Assuming $a_{s+1} = d_s$ and $x_{s+1} = 0$, for each job j_i the following steps are performed;

i) Calculate the overlap parameter O_i using the following equation;
$$O_i = max(0, d_i - a_{i+1} - B(a_{i+1}, d_i)) \tag{5}$$
ii) Calculate the flexibility coefficient x_i using the following equation;
$$x_i = d_i - a_i - C_i - B(a_i, d_i) - O_i + min(x_{i+1}, O_i) \tag{6}$$

2) Online Phase: After finishing the offline phase, all the parameters for each job and the partition blockings V_p are passed to the online scheduler. The online scheduler is activated at each job activation time a, finish time f, and when the partition is active *and* the processor is idle at the release of a new aperiodic job.

When the scheduler is activated at time t and an aperiodic job release is detected, the guarantee test is performed. Similar to the flat scheduling model, the job j_n is defined as the next job to be activated from the scheduling table S_p, while the job j_l is defined as the first job activated after the deadline of the released aperiodic job j_{ap}. During the guarantee test, the aperiodic room R_i for each job i from j_n to j_l is calculated using the following set of equations;

$$\begin{aligned} s_i &= \begin{cases} f_{i-1}, & n < i < l \\ t, & i = n \end{cases} \\ R_i &= a_i - s_i - B(s_i, a_i) + \\ &\quad min(d_{ap} - a_i - B(a_i, d_{ap}), x_i) \end{aligned} \tag{7}$$

The guarantee test passes if, for any job $j_i | n \leq i < l$, the aperiodic room R_i is larger than or equal to C_{ap}. Once the guarantee test passes, the guarantee algorithm is activated. The guarantee algorithm requires the following steps to be performed in the defined order:

i) Insert the released aperiodic job j_{ap} in the schedule S_p before job j_i , i.e. $a_i = a_{ap} = s_{i+1}$ and $f_i = f_{ap} = A(a_{ap} + C_{ap})$.
ii) For each job j_k, such that $k > i \wedge a'_k > a_k$, perform the following steps starting from j_{i+1};
   a) $a'_k = A(a_k + max(f_{k-1} - a_k, 0))$
   b) $f_k = A(f_k + (a'_k - a_k - B(a_k, a'_k)))$
   c) $x_k = x_k - (a'_k - a_k - B(a_k, a'_k))$
   d) $a_k = a'_k$
iii) For each job j_p, such that $p \leq i$, calculate x_p similar to the offline phase starting from j_i and ending at j_n. The process of modifying x_p stops when the old x_p is equal to the new x_p.

During run-time, the jobs are scheduled similar to the flat scheduling model.

D. Job-Shifting Example

Offline Phase: Consider two jobs j_1 and j_2 with release time r, activation time a, finish time f, deadline d, and worst-case execution time C as defined in Table II. Assume that the partition blocking set V is defined to be $\{\langle 5, 8\rangle\}$ as shown in Fig. 2. During the offline phase of job-shifting algorithm, the

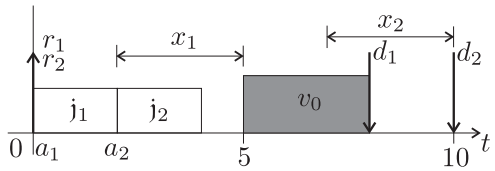


Fig. 2: Example for offline phase of job-shifting

j	r	a	f	d	C	x
1	0	0	2	8	2	**3**
2	0	2	4	10	2	**3**

TABLE II: Example parameters after offline phase

j	r	a	f	d	C	x
0	0	**0**	**2**	3	2	**1**
1	0	**2**	**4**	8	2	**1**
2	0	**4**	**9**	10	2	**1**

TABLE III: Example parameters after online phase

flexibility coefficients x are calculated starting from job j_2 and ending at job j_1 as under;

For job j_2, $a_3 = d_2$ and $x_3 = 0$.

$$\begin{aligned} O_2 &= max(0, d_2 - a_3 - B(a_3, d_2)) \\ &= max(0, d_2 - d_2 - B(d_2, d_2)) = 0 \\ x_2 &= d_2 - a_2 - C_2 - B(a_2, d_2) - O_2 + min(x_3, O_2) \\ &= 10 - 2 - 2 - 3 - 0 + min(0, 0) = 3 \end{aligned}$$

For job j_1, $a_2 = 2$ and $x_2 = 3$.

$$\begin{aligned} O_1 &= max(0, d_1 - a_2 - B(a_2, d_1)) \\ O_1 &= max(0, 8 - 2 - 3) = 3 \\ x_1 &= d_1 - a_1 - C_1 - B(a_1, d_1) - O_1 + min(x_2, O_1) \\ &= 8 - 0 - 2 - 3 - 6 + min(6, 6) = 3 \end{aligned}$$

Online Phase: Assume an aperiodic job j_{ap} with $C_{ap} = 2$ and $d_{ap} = 3$ is released at time $t = 0$ in the example shown in Fig. 2. The online scheduler performs the guarantee algorithm with $n = 1$ and $l = 3$ (i.e. first job of the next SC). For job j_i with $i = 1$, $s_1 = t = 0$. The aperiodic room R_1 is obtained to be $R_1 = a_1 - s_1 + min(d_{ap} - a_1 - B(a_1, d_{ap}), x_1) - B(a_1, s_1) = 0-0+min(3-0-0, 3)-0 = 3$. Since $R_1 > C_{ap}$, the newly released job j_{ap} can be guaranteed to finish before its deadline d_{ap}. Therefore, the guarantee algorithm is triggered. The steps are shown below;

i) Job j_{ap} is inserted at location 0, i.e. $a_0 = a_{ap} = s_1 = 0$, $f_0 = f_{ap} = A(a_{ap} + C_{ap}) = 2$, $C_0 = C_{ap} = 2$ and $d_0 = d_{ap} = 3$.

ii) For job j_1:

$$\begin{aligned} a'_1 &= A(a_1 + max(f_0 - a_1, 0)) \\ &= A(0 + max(2 - 0, 0)) = 2 \\ f_1 &= A(f_1 + (a'_1 - a_1 - B(a_1, a'_1)) \\ &= A(2 + (2 - 0 - B(0, 2))) = 4 \\ x_1 &= x_1 - (a'_1 - a_1 - B(a_1, a'_1)) \\ &= 3 - (2 - 0 - B(0, 2)) = 1 \\ a_1 &= a'_1 = 2 \end{aligned}$$

For job j_2:

$$\begin{aligned} a'_2 &= A(a_2 + max(f_1 - a_2, 0)) \\ &= A(2 + max(4 - 2, 0)) = 4 \\ f_2 &= A(f_2 + (a'_2 - a_2 - B(a_2, a'_2)) \\ &= A(4 + (4 - 2 - B(2, 4))) = 9 \\ x_2 &= x_2 - (a'_2 - a_2 - B(a_2, a'_2)) \\ &= 3 - (4 - 2 - B(2, 4)) = 1 \\ a_2 &= a'_2 = 4 \end{aligned}$$

iii) For job j_0:

$$\begin{aligned} O_0 &= max(0, d_0 - a_1 - B(a_1, d_0)) \\ &= max(0, 3 - 2 - B(2, 3)) = 1 \\ x_0 &= d_0 - a_0 - C_0 - B(a_0, d_0) - O_0 + min(x_1, O_0) \\ &= 3 - 0 - 2 - B(0, 3) - 1 + min(1, 1) = 1 \end{aligned}$$

The modified scheduling table parameters are shown in Table III. After performing the guarantee test, the job j_0 is selected for execution since $a_0 = t = 0$ and the best-effort queue is empty. When the job j_0 completes, the scheduler gets activated again at $t = 2$ and schedules job j_1, and then at $t = 4$, the scheduler executes job j_2.

E. Mixed-Critical Tasks

It is important to note that neither flat nor hierarchical scheduling models take the task criticality Y into account. The reason for such a deliberate elimination is the use of the industrial mixed-criticality model defined by the standards IEC61508 [17], DO-178C [18] and ISO26262 [19]. In these standards, the task criticality Y is used to refer to the level of assurance applied to the development of software application and the different criticality tasks are segregated by allocating them to different partitions [1]. For such industrial standards, the tasks only define single WCET C, the higher criticality level of a task does not mean greater importance and, therefore, does not warrant rejection of lower criticality tasks. All these facts lead to a conclusion that the task criticality cannot be exploited by the scheduler (as long as critical and non-critical tasks stay in separate partitions). For more information on industrial mixed-criticality model, please refer to [1].

F. Offline Guarantee Analysis

In order to design a robust and responsive real-time system, a usual practice is to guarantee the correct behaviour of a sub-system or task offline. To guarantee some service to aperiodic activities, a certain amount of processor bandwidth is usually reserved for aperiodic tasks (or servers). Due to the limited system resources and large number of constraints, there is always a limit to the bandwidth which is considered a 'safe reservation' for handling aperiodic activities. In industry, such reservations are considered mandatory to guarantee system safety. However, reserving a specific bandwidth for such a case still limits the service for aperiodic execution. Moreover, the response-time of aperiodic tasks in such reservations also has a strong dependency on the employed reservation technique. In this work, we conjecture that, instead of reserving a limited amount of processor bandwidth, all free partition/processor bandwidth can be utilized to service aperiodic tasks. The following theorem provides the base for our conjecture:

Theorem IV.1. For a given TT schedule S, if there exists a reservation such that a number of non-preemptive aperiodic jobs with a defined job release pattern can be executed feasibly, the job-shifting algorithm can also execute them without a reservation.

Proof. Consider a TT scheduling table S with defined activation times a_i and finish times f_i for each periodic job j_i. A reservation window $\Lambda_{[t_1,t_2)}$ with window start time t_1 and end time t_2 is defined as contiguous *reserved* processing time for the execution of non-preemptive aperiodic job(s). According to Definition IV.2, the aperiodic room R_k is defined as the *maximum* room for executing non-preemptive aperiodic job, where $k \in \mathbb{N} | f_{k-1} \leq t_1 \wedge a_k \geq t_2$. When a non-preemptive aperiodic job j_{ap} with $C_{ap} = t_2 - t_1$ is released at $r_{ap} \leq t_1$, then by definition, the condition $\Lambda_{[t_1,t_2)} \leq R_k$ always holds. If the non-preemptive aperiodic job j_{ap} can be feasibly executed by $\Lambda_{[t_1,t_2)}$, R_k can also feasibly execute it without an offline reservation. The proof can be iteratively applied to multiple reservation windows. Hence, the theorem is proven. □

In other words, the same practice of 'safe reservation' can be utilized to give guarantees for aperiodic execution in job-shifting without any *online* reservation. However with job-shifting, the bandwidth loss due to partitioning [2] can be significantly reduced (further in Sec. V) and the aperiodic response-times can be improved (further in Sec. IV-G2).

Being a non-preemptive algorithm, a necessary condition for aperiodic guarantee can also be checked offline in job-shifting. Independent of the release time of the aperiodic job, there must exist enough aperiodic room R_i in the schedule S_p to accommodate complete aperiodic job. For an aperiodic task, if there exists no aperiodic room $R_i | R_i \geq C_{ap}$ for any $\mathrm{j}_i \in S_p$, the aperiodic job will never be able to execute while keeping feasibility of other tasks in S_p. This necessary condition is checked between the offline and the online phases of the job-shifting algorithm.

G. Discussion

In this section, we provide a description of how the job-shifting algorithm can be optimized and which factors can increase the online overheads.

1) Hypervisors and Clocks: There exists a multitude of real-time hypervisors in the market today. Based on the type of interface they provide to the partition, the online complexity of the job-shifting algorithm can be modified. Some hypervisors, e.g. XtratuM [22], provide a partition local clock which ticks only when the partition is active. In such hypervisors, the equations for the online phase of the flat scheduling model can be used in a hierarchic design by defining a_i, r_i and d_i in terms of the partition local clock. For such an implementation, the adjust operator $A(u)$ can be completely eliminated and the blocking operator $B(p, q)$ is only required for the last step of guarantee algorithm (where the flexibility coefficient is calculated). However, there exist hypervisors which do not provide a partition local clock, e.g. PikeOS [23], and therefore need to implement both operators $A(u)$ and $B(p, q)$.

2) Feasibility & Complexity: When the scheduler needs to adjust a number of aperiodic jobs, the problem of guaranteeing all the aperiodic jobs becomes a variant of bin-packing problem. The decision version of the bin-packing problem is known to be NP-Complete [24]. Moreover, a non-clairvoyant online scheduler, by definition, cannot guarantee that all randomly released aperiodic jobs can be accepted.

For a single aperiodic job, the *maximum* number of checks (i.e. $R_i \geq C_{ap}$) performed by the guarantee test are equal to the number of guaranteed jobs activated during the interval $[t, d_{ap})$, which is not different from the original slot-shifting algorithm [4]. Depending on the required performance metric, the guarantee test can utilize any bin-packing heuristic, e.g. first-fit, best-fit or worst-fit. The first-fit heuristic can be used when least aperiodic response-time and online overheads are required. In such a heuristic, the guarantee test starts from job j_n and inserts the newly released aperiodic job before the first job with enough aperiodic room R_i. On the contrary, when larger online overheads can be tolerated but better distribution of aperiodic jobs is required, the best- or worst-fit heuristics can be utilized. Moreover, the order in which multiple aperiodic jobs are guaranteed also impacts the overall feasibility of jobs. Lupu et al. [25] recommend worst-fit decreasing utilization heuristic for scheduling.

For the flat scheduling model, the guarantee test and the guarantee algorithm have a complexity of $O(m)$, where m is the number of jobs activated during the interval $[t, d_{ap})$ (or $l - n$). For the hierarchic scheduling model, we introduced two operators $A(u)$ and $B(p, q)$. A naïve implementation of these operators has a complexity of $O(r)$, where r is the cardinality of the blocking set V. When partition local clock is available, the complexity of the guarantee test does not change. However, the complexity of the guarantee algorithm changes to $O(mr)$. On the contrary, when the partition local clock is not available, the complexity of the guarantee test becomes $O(mr)$, while the complexity of the guarantee algorithm changes to $O(mr^3)$.

3) Optimizations: At the start of this section, it was assumed that all the tasks execute for their complete worst-case execution time C. However, this assumption seldom holds during system operation. A task executing more than C time can lead to the violation of the temporal isolation between tasks of the same application. In the worst case, such violations may never lead to a processor yield for other tasks to execute. To protect from such a situation, a hardware timer can be programmed to generate an overrun interrupt which terminates the misbehaving job and returns the control to the scheduler. When the job executes less than C time, the overrun interrupt can be terminated (or rescheduled for the next job). In such a situation, the free processing time can be utilized by using a simple approach. At the observed job finish time t, if for the next job $a_n > r_n$, the flexibility coefficient x_n can be updated to $x_n + (a_n - max(t, r_n))$ and the job can be preponed to activate at $max(t, r_n)$. If for the next job $a_n = r_n$, the free resources can be accounted for by the aperiodic room R_n and therefore, require no further action.

When a large number of aperiodic jobs are released before the scheduler activation, performing guarantee algorithm for all of them may lead to large scheduler overheads. In the worst case, the reserved processing time to execute the next job might be reduced, leading to a job incompletion. To avoid such a scenario, Schorr [6] proposed a heuristic where at most one aperiodic job is guaranteed for each activation of the scheduler.

4) Intra-Partition Scheduling: In job-shifting algorithm, the degree of flexibility of a job is defined by the flexibility coefficient. If the flexibility coefficient for all the jobs is zero, the job-shifting algorithm reduces to the background processing approach. Therefore, it can be said that the flexibility coefficient is the direct measure of the adaptability of a schedule. Depending on the scheduler used prior to the offline phase of job-shifting, the flexibility coefficient can vary significantly. The EDF approach results in the largest flexibility coefficients, while the latest-deadline-first (LDF) results in the least.

In Sec. III, the input for the offline phase of job-shifting was defined to be a feasible TT partition schedule S_p. However, this restriction can be relaxed when (i) the online scheduling strategy is known *and* (ii) it can be made sure that the estimated order of execution of jobs will not be violated online. For such an online scheduler, the flexibility coefficient for all the jobs can be calculated offline. Nonetheless, the online scheduling of non-preemptive periodic tasks is an NP-Hard problem [11] even with harmonic periods [26] or with arbitrary period ratios [27]. Moreover, the non-preemptive versions of EDF or RM are not optimal, even when the online scheduler is work-conserving or non-work-conserving [27]. Due to the complexity of the problem, a feasible TT partition schedule S_p seems a better choice as an offline exhaustive search can be done (perhaps with large search time) with minor efforts to generate such a schedule.

V. Efficiency Evaluation

In this section, we present the results of the experiments performed to evaluate efficiency of the job-shifting algorithm.

A. Experimental Setup

In order to provide a reference point for the job execution guarantees provided by the job-shifting (JS) algorithm, we implemented a non-preemptive background scheduler (NP-BG). The NP-BG scheduler services the aperiodic jobs during the idle time available in the scheduling table, i.e. a non-preemptive aperiodic job is executed only when it can be guaranteed that the job will finish within the idle time. Note that any existing algorithm cannot be used to provide the reference point for comparison due to varying task models.

For the evaluation of job-shifting algorithm, 1000 task-sets were generated. The parameter ranges for generating task-sets were selected as defined by Schorr [6] to capture the behaviour of real applications. Each generated task-set consisted of at most one processor and one partition. The partition blockings V are generated with a strict period of 10 and a relative beginning time $b = 6$ of each blocking $v \in V$ until the end of the scheduling cycle SC.

Inside the partition, [1, 3] periodic tasks were generated with phases $\phi = 0$, WCET C in the interval [1, 15] and the period T in the interval [15, 30] (with implicit deadlines). The total utilization of periodic tasks was kept 25% and the task parameters were generated using UUniFast [28] algorithm to get uniform tasks distribution. The aperiodic tasks were generated with release time r in the interval [0, SC), the WCET C in the interval [5, 10] and the deadline d = DLX $\times C$, where DLX is the deadline extension factor. The factor DLX defines the tightness of the deadline compared to C. The generation process of the aperiodic tasks is stopped when the aperiodic utilization reaches the target utilization.

To distinguish the effects of varying urgencies, task-sets were generated for each different DLX factor in the list {4, 8, 12} and the aperiodic tasks utilization in the list {5%, 10%, 15%, 20%}. The guarantee ratios (i.e. the ratio of the number of guaranteed aperiodic jobs to the total number of aperiodic jobs) of the algorithms are also affected by the partition supply. Therefore, all the task-sets were generated for two partition utilizations in the set {70%, 50%}.

As mentioned in Sec. IV, the job-shifting (JS) algorithm requires a cyclic executive schedule for the partition. Therefore during the pre-processing stage, the partition schedule S_p is generated using the EDF scheduler. The task-sets which resulted in SC lengths outside of the interval [500, 5000) were rejected since the real-world tables are smaller [6]. Moreover, the task-sets were also filtered using the offline guarantee analysis method mentioned in Sec. IV-F.

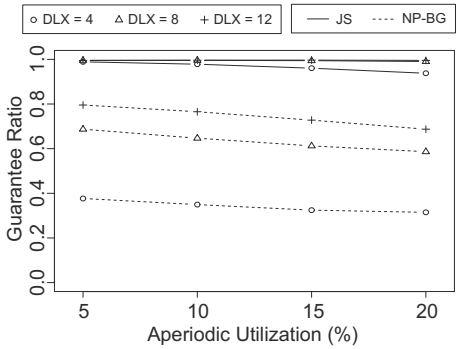
B. Results & Discussion

Fig. 3 shows the average guarantee ratio for the generated task-sets (i.e. each point in the figure represents the average guarantee ratio of 1000 task-sets) for varying aperiodic task utilizations. In the figure, the DLX factor is represented with different pointer types and the schedulers (JS and NP-BG) with different line types. For the non-preemptive background scheduler (NP-BG), the guarantee ratio defines the ratio of the number of aperiodic jobs which finished before their deadlines to the total number of aperiodic jobs.

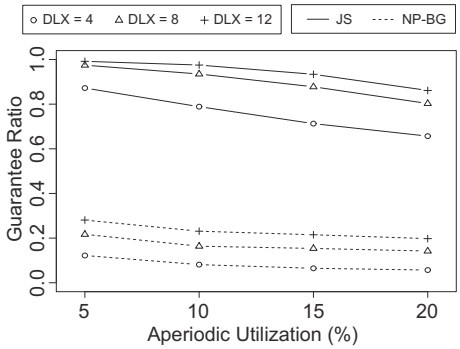
Fig. 3 shows that, for the background scheduling, the number of finished aperiodic jobs decreases to a great extent (e.g. from 0.7 to 0.2 for DLX=12 and aperiodic utilization U_{ap} = 20%) due to just 20% decrement in the partition supply. However, for the job-shifting algorithm, the number of finished aperiodic jobs did not suffer as much (e.g. from 1.0 to 0.9 for DLX=12 and aperiodic utilization U_{ap} = 20%). Moreover, notice in Fig. 3 that the average guarantee ratio of the job-shifting algorithm is quite large compared to the background processing. Therefore, we conclude that job-shifting is more suitable for online aperiodic job admission (at least, in comparison to the background processing).

VI. Overheads Evaluation

In this section, we present the results of the experiments performed to evaluate overheads incurred by the job-shifting



(a) 70% Partition supply



(b) 50% Partition supply

Fig. 3: Comparison of job-shifting with background processing

algorithm. In Sec. VI-A, we provide the description of the experimental setup. While in Sec. VI-B, we present and briefly discuss the obtained results.

A. Experimental Setup

In order to evaluate the job-shifting overheads, we implemented the job-shifting algorithm in the DREAMS project[13]. In the following, we provide a description of the hardware and software platforms, one of the safety critical demonstrator and the job-shifting integration strategy in DREAMS.

1) Hardware & Software Platforms: The DREAMS Harmonized Platform (DHP) is a heterogeneous multi-core platform developed on top of the Xilinx Zynq ZC-706 FPGA platform. The Zynq platform provides a dual-core ARM Cortex-A9 processor running at 400Mhz. The DHP design extends the ARM processor with 3 MicroBlaze cores. The different cores are interconnected through a dedicated network designed to provide support for time critical communication.

In the context of the DREAMS project, the XtratuM hypervisor [22] was ported to run on top of the ARM processor, Freescale QorIQ T4240 and Intel x86. Moreover, a layer is added between the XtratuM and the user applications to provide services like local and global resource management with support for online system reconfiguration [29]. To implement these services three special system partitions were developed: the monitor (MON), the local resource manager (LRM) and the global resource manager (GRM). Additionally, the user partitions (on which the applications are deployed) implement a local resource scheduler (LRS) which provides the applications with a cyclic-executive intra-partition scheduler (CEIPS [20]) and an interface to the developed services.

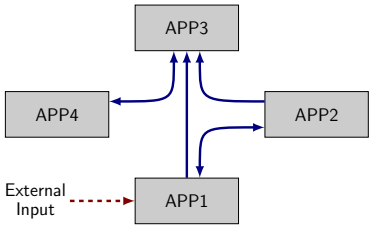


Fig. 4: Demonstrator communications (solid arrows) and external inputs (dashed arrow)

Application	# Periodic Tasks	U_p	# Aperiodic Tasks	
			Comm	Ext. Input
APP1	6	0.021	1	17
APP2	10	0.545	1	0
APP3	5	0.055	3	0
APP4	1	0.015	0	0
Total	22	0.636	5	17

TABLE IV: Tasks for the safety critical demonstrator

2) Safety Critical Demonstrator: In the context of DREAMS, a demonstrator of a safety critical system was developed by Thales to assess the technologies developed within the project. The demonstrator can be seen as a distributed system composed of three platforms (2 Freescale QorIQ T4240 and a DHP) interconnected via a TTEthernet switch [29].

The safety critical demonstrator is composed of four different applications (APP1–APP4) communicating with each other as shown in Fig. 4. Each application consists of a number of periodic and aperiodic tasks as defined in Table IV. The tasks sending a message to other task(s) are executed synchronously (i.e. periodic tasks), during which they decide if data needs to be communicated; while the tasks receiving messages from other task(s) are executed asynchronously[1] (i.e. aperiodic tasks) when data is received on a defined virtual network port. Apart from the communicating tasks, there exist a number of tasks which are activated by external events, e.g. push of a button. Fig. 4 shows that a number of tasks in the application APP1 are triggered by external inputs (see Table IV) which are managed by aperiodic tasks.

3) Job-Shifting Integration: In order to generate the required input for partition p, which utilizes the job-shifting algorithm, a tool from DREAMS tool-chain called MCOSF (Mixed-Criticality Offline Scheduling Framework) is used. The MCOSF tool calculates the flexibility coefficients x for each job of partition p from the application, platform and system software model defined by the DREAMS meta-model [30]. The generated information is forwarded to the Local Resource Scheduler (LRS [31]) of partition p.

In this work, the LRS for the partitions is modified to implement the job-shifting algorithm with dense time-base [8] (see

[1]With the exception of the communication from APP3 to APP4, to evaluate overheads for partitions without any aperiodic task.

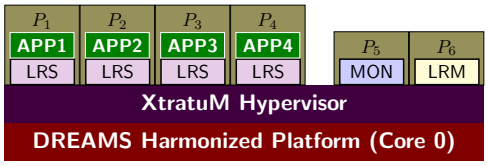


Fig. 5: Demonstrator deployment

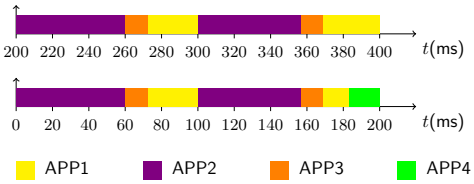


Fig. 6: Hypervisor schedule for safety critical demonstrator

Application	Flex. Coeff. (x)		Act. Probability	
	min (ms)	max (ms)	min	max
APP1	13	30	0.008	0.35
APP2	0	4	NA	NA
APP3	0	7	NA	NA
APP4	0	2	NA	NA

TABLE V: Application deployment parameters

Sec. III) on top of CEIPS (see Sec. VI-A1). As CEIPS does not make use of a ready-queue, a simple job dispatch table for each partition slot [22] is specified in the Partition Configuration File (PCF). To detect the aperiodic job activation, the sources of the aperiodic task activation are also specified in the PCF (i.e. virtual network ports for message activated tasks, and interrupt sources for tasks activated due to external inputs). As the job dispatch table is implemented using a contiguous array, the memory required for accommodating new aperiodic jobs is statically reserved. Moreover, similar to the heuristic proposed by Schorr [6], at most one aperiodic job is guaranteed per LRS activation in order to bound the scheduler overheads.

Besides the constraints mentioned in Sec. III, the scheduler CEIPS puts forward another constraint; A job started in a partition slot s_p [22] must finish before the end of s_p. To accommodate this constraint, each partition blocking $v \in V$ (i.e. the opposite of partition slots in XtratuM [22]) can be considered a job for the guarantee test/algorithm (with $a = b, C = m - b, f = m$ and $x = 0$). The advantage of such an assumption enables the use of flat scheduling model for the implementation in the LRS, in turn reducing the run-time complexity of the job-shifting algorithm (See Sec. IV-G2).

4) System Deployment: While the DHP (see Sec. VI-A1) provides multiple cores, only one of the ARM cores running the DREAMS extended XtratuM hypervisor was used for the overhead evaluation. This approach enables evaluation with large enough core utilization, further exhibiting benefits of the job-shifting algorithm over reservation based or background approaches. Moreover, other DREAMS services (e.g. global resource management) were disabled as their focus is orthogonal to the overheads evaluation of the job-shifting algorithm.

For the overheads evaluation of the job-shifting algorithm, the four applications mentioned in Sec. VI-A2 were deployed each in its own user partition on a single ARM core of the DHP. Moreover, the external inputs to activate aperiodic tasks were triggered with a defined probability to simulate event occurrence. This process enables more frequent aperiodic activations compared to generating the events manually or through measurements in the environment. The system was deployed as shown in Fig. 5, including the previously described LRM and MON system partitions. Due to large range of the task periods, the generated schedule resulted in a scheduling cycle length SC = 40s. To give a rough idea, the initial part of the generated hypervisor schedule is shown in Fig. 6, while the range of flexibility coefficients and the activation probabilities for tasks activated by external events are provided in Table V. In the table, the activation probability of 1 means that a job of the aperiodic task is released every second.

B. Results & Discussion

Fig. 7 shows the bar plot of the measured maximum overheads for each partition utilizing job-shifting scheduler within 1000 scheduling cycles SC (See Sec. III). The error bars in Fig. 7 represent the 98% confidence interval for the cumulative overheads. The legends in Fig. 7 represent the overheads distribution, where 'Queue Shifting' overheads represent the time elapsed in inserting a newly released aperiodic job in the job dispatch table and the 'Rest Overheads' represents the rest of the scheduling overheads (e.g. next job selection). It is important to mention here that the graph in Fig. 7 includes the logging overheads and excludes the overheads due to the detection of aperiodic task arrival, since these overheads are dependent on the aperiodic activation source, the hypercalls implementation, and the platform architecture. For the aperiodic tasks activated by the reception of a message on a virtual network port, the overheads were measured to be approximately 13μs per aperiodic task. Whereas the detection of the aperiodic tasks triggered by the external events required approximately 2μs per aperiodic task.

Fig. 7 shows that the overheads incurred by the job-shifting algorithm strongly depend on the number of tasks inside a partition. Moreover, the overheads of the guarantee test and the guarantee algorithm depend on the number of periodic tasks, the number of aperiodic tasks and the flexibility of guaranteed tasks inside a partition. It is important to mention here that the overheads due to queue shifting can be significantly reduced

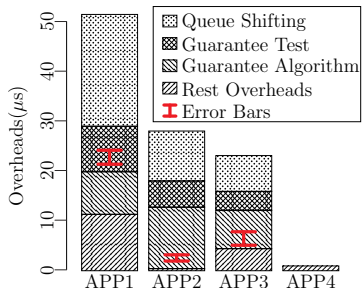


Fig. 7: Job-Shifting overheads on Zynq board (Error bars represent 98% confidence intervals for cumulative overheads)

by implementing the schedule using a double linked list. Since the V&V efforts for such a modification in a legacy partition data structure are higher, it is avoided in this evaluation.

It was also observed that, on average, 22.6 aperiodic tasks were released per second during the execution of 1000 scheduling cycles SC. Furthermore, none of the released aperiodic jobs missed its deadline and all the aperiodic jobs were guaranteed for the first invocation of Equation 4 due to relatively smaller utilizations of periodic tasks and good enough aperiodic room R.

In 2015, Schorr [6] measured the overheads of the slot-shifting algorithm on a cycle accurate MPARM simulator running at 200MHz. He mentioned that, for his large set of synthetic task-sets, the overheads for slot-shifting acceptance test ranged from 8.695μs to 23.7μs, while the overheads for the guarantee algorithm ranged from 9.82μs to 42.99μs (excluding queue shifting overheads, see experiment 3 in [6]). For our safety critical demonstrator application, Fig. 7 shows that the overheads of job-shifting algorithm are comparatively smaller (i.e. 3.825μs to 9.17μs for guarantee test and 7.71μs to 12.3775μs for guarantee algorithm). The smaller overheads for job-shifting on the Zynq ZC706 platform were partly expected as the operating frequency of the platform is twice compared to the MPARM. However, this comparison manifests that the overheads incurred by the job-shifting algorithm are quite comparable to preemptive algorithms (e.g. slot-shifting) despite the NP-hard nature of non-preemptive scheduling problem (see Sec. IV-G2).

VII. Conclusion

Modern industrial applications are required to incorporate ever increasing number of features. To reduce the verification and validation costs, hypervisors are utilized to partition the resources among the applications. One of the major problems with partitioning is loss of bandwidth [2], which can significantly reduce the effective processor utilization when reservation mechanisms are used for servicing aperiodic task.

In this paper, we presented an algorithm for online admission of non-preemptive aperiodic tasks in hierarchical systems. We also provided the basis of a necessary test for the admission of non-preemptive aperiodic tasks. We evaluated the efficiency of our algorithm on synthetic but practical task-sets and demonstrated that our approach efficiently utilizes the available resources. Furthermore, we also implemented our algorithm on an ARM based platform. The experiments manifested that the overheads incurred by our scheduler are very small and practical for safety critical applications.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7/2007–2013 under grant agreement n°610640.

References

[1] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *International Conference on Real-Time and Networks Systems (RTNS), 2015*.

[2] A. Lackorzyński, A. Warg, M. Völp, and H. Härtig, "Flattening hierarchical scheduling," in *ACM International Conference on Embedded Software (EMSOFT), 2012*.

[3] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," in *IEEE Conference on Real-Time Systems Symposium (RTSS), 1992*.

[4] G. Fohler, "Flexibility in statically scheduled real-time systems," Ph.D. dissertation, TNF, Wien, Österreich, April 1994.

[5] D. Isović and G. Fohler, "Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints," in *IEEE Conference on Real-time Systems Symposium (RTSS), 2000*.

[6] S. Schorr, "Adaptive real-time scheduling and resource management on multicore architectures," Ph.D. dissertation, March 2015.

[7] J. Theis, "Certification-Cognizant Mixed-Criticality Scheduling in Time-Triggered Systems," Ph.D. dissertation, March 2015.

[8] H. Kopetz, "Sparse Time versus Dense Time in Distributed Real-Time Systems," in *International Conference on Distributed Computing Systems (ICDCS), 1992*.

[9] M. Nasri, S. Baruah, G. Fohler, and M. Kargahi, "On the optimality of RM and EDF for non-preemptive real-time harmonic tasks," in *International Conference on Real-Time Networks and Systems, 2014*.

[10] H. Ramaprasad and F. Mueller, "Bounding Worst-Case Response Time for Tasks with Non-Preemptive Regions," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2008*.

[11] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *IEEE Conference on Real-time Systems Symposium (RTSS), 1991*.

[12] Xilinx, http://www.xilinx.com/, retrieved: 10th Apr, 2017.

[13] "DREAMS Project," http://dreams-project.eu/, retrieved: 10th Apr, 2017.

[14] S. R. Thuel and J. P. Lehoczky, "Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing," in *IEEE Conference on Real-Time Systems Symposium (RTSS), 1994*.

[15] T.-S. Tia, J. W. S. Liu, and M. Shankar, "Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems," *Real-Time Systems*, 1996.

[16] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *IEEE Conference on Real-Time Systems Symposium (RTSS), 2007*.

[17] IEC61508, "Functional safety of electrical/electronic/programmable electronic safety-related systems," IEC, Standard, 2010.

[18] DO-178C, "Software Considerations in Airborne Systems and Equipment Certification. RTCA," Inc., Standard, 2011.

[19] ISO26262, "Road vehicles - Functional Safety," ISO, Standard, 2011.

[20] ARINC-653, "Avionics Application Software Standard Interface," ARINC Inc., Standard, 2003.

[21] J. Y.-T. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information processing letters*, 1980.

[22] A. Crespo, I. Ripoll, M. Masmano, P. Arberet, and J. Metge, "Xtratum an open source hypervisor for tsp embedded systems in aerospace," *Data Systems In Aerospace (DASIA), 2009*.

[23] PikeOS, https://www.sysgo.com/products/pikeos-hypervisor/, retrieved: 10th Apr, 2017.

[24] E. G. Coffman, Jr, M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, no. 1, pp. 1–17, 1978.

[25] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA), 2010*.

[26] Y. Cai and M. Kong, "Nonpreemptive scheduling of periodic tasks in uni-and multiprocessor systems," *Algorithmica*, vol. 15, no. 6, 1996.

[27] M. Nasri and G. Fohler, "Non-Work-Conserving Non-Preemptive Scheduling: Motivations, Challenges, and Potential Solutions," in *Euromicro Conference on Real-Time Systems (ECRTS), 2016*.

[28] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[29] G. Durrieu, G. Fohler, G. Gala, S. Girbal, D. Gracia Pérez, E. Noulard, C. Pagetti, and S. Pérez, "DREAMS about reconfiguration and adaptation in avionics," in *ERTS 2016*, 2016.

[30] DREAMS consortium, "Meta-models for Application and Platform," D1.4.1, Mar. 2015.

[31] T. Koller, G. Gala, D. Gracia Pérez, C. Ruland, and G. Fohler, "DREAMS: Secure Communication Between Resource Management Components in Networked Multi-Core Systems." IEEE Conference on Open Systems, 2016.