View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CiteSeerX

Limited Pre-emptive Global Fixed Task Priority

José Marinho* Vincent Nélis* Stefan M. Petters* Marko Bertogna† Robert I. Davis ‡
*CISTER/INESC-TEC, Polytechnic Institute of Porto, Portugal
†University of Modena, Modena, Italy
‡University of York, York, UK
Email: {jmsm,nelis,smp}@isep.ipp.pt, marko.bertogna@unimore.it, rob.davis@york.ac.uk

***Abstract*—In this paper a limited pre-emptive global fixed task priority scheduling policy for multiprocessors is presented. This scheduling policy is a generalization of global fully pre-emptive and non-pre-emptive fixed task priority policies for platforms with at least two homogeneous processors. The scheduling protocol devised is such that a job can only be blocked at most once by a body of lower priority non-pre-emptive workload. The presented policy dominates both fully pre-emptive and fully non-pre-emptive with respect to schedulability. A sufficient schedulability test is presented for this policy. Several approaches to estimate the blocking generated by lower priority non-pre-emptive regions are presented. As a last contribution it is experimentally shown that, on the average case, the number of pre-emptions observed in a schedule are drastically reduced in comparison to global fully pre-emptive scheduling.**

I. Introduction

The drive enhancing computational throughput has changed its focus from increasing transistor switch frequency to replicating functional units. As frequency of operation increases the ability for memory to feed data to the faster cores cannot keep up [1]. The discrepancy between memory and processor throughput is not the only limiting factor. The power dissipated in faster clocked processors also becomes prohibitive both in terms of energy wasted and required energy dissipation mechanisms [2]. The response of academia and industry was to replicate the cores instead of increasing the clock frequency. Throughput gains are nevertheless not linearly proportional to the number of cores.

As multicores are currently a mainstream computing apparatus, and will remain so in the foreseeable future, some COTS platforms are being increasingly adopted as target platforms in the embedded domain. In time it is expected that these will be used in the high criticality embedded world. It is thus important to study the scheduling theory that would govern the operation of such systems.

Current real-time operating systems provide support for symmetric multiprocessor scheduling. A number of global scheduling policy types exist. These can be categorized into distinct classes with respect to where tasks and their constituent jobs are allowed to execute [3]. One extreme is the fully partitioned scheduling. Tasks are statically allocated to one processor, its workload can then only execute on the same single processor. On the other end of the spectrum lies the global scheduling where there is no *a-priori* restriction on which processor the workload of any task will execute. Both fully partitioned and global fixed task priority scheduling policies are provided out of the box in popular real-time operating systems [4].

A shortcoming of the literature on the schedulability assessment for global multicore scheduling is that the workload is assumed not to incur additional overheads when pre-emptions and migrations happen. The cost of pre-emptions and especially migrations between cores that do not share cache have been shown to be quite significant [5]. As in fully pre-emptive disciplines for single core it is difficult to quantify the number of pre-emptions a given task is subject to. Moreover in multicore when a pre-emption occurs a subsequent migration may take place. It is then beneficial to quantify these events and, if possible, avoid these effects when these prove unnecessary for correct system temporal behaviour. With this intent in mind we sought to devise the theory of limited pre-emptive scheduling, now somewhat mature in single core, in symmetric multicore. The model presented allows for the accurate definition of a limited set of points where a given task may be pre-empted, hence allowing for reduced pessimism when analysing the effect of pre-emptions and migrations.

In this paper we address a very particular global scheduling discipline denominated global fixed task priority with fixed non-pre-emptive regions. This scheduling policy is work-conserving. This means that when workload is available and it is not being executed on any of the processors it is the case that all processors are currently executing some other workload. The fixed task priority term refers to the fact that each task has a base priority. When workload belonging to a task is executed on the platform it does so with the task base priority. By fixed non-pre-emptive regions it is implied that each job is composed of a set of sub jobs which execute without pre-emption. When a sub-job commences execution on a processor it cannot be pre-empted until it terminates. A task can then only be pre-empted at sub-job boundaries which are referred to as pre-emption points. Pre-emption points may be implemented either through interrupt disabling or even via a system call indicating the start of a non-pre-emptive region to the system dispatcher which acts accordingly.

The particularity of the proposed solution is such that some job of a given task can be blocked at most once by lower priority workload before it is first dispatched. In this paper we provide a sufficient schedulability test for this scheduling model. In this introductory work we assume that the pre-emption and migration delays have negligible cost. This allows for a clearer explanation and discussion of the basic limited pre-emptive concepts.

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within REPOMUC project, ref. FCOMP-01-0124-FEDER-015050, by National Funds through FCT (Portuguese Foundation for Science and Technology) and by the EU ARTEMIS JU funding, within CONCERTO project, ref. ARTEMIS/0003/2012, JU grant nr.333053 and by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/81085/2011.

II. System Model

In this paper we consider the workload to be modeled as a task-set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ composed of n tasks. Each task is characterized by the four-tuple $\langle C_i, D_i, T_i, Q_i \rangle$. The parameter C_i represents the worst-case execution time of each job from τ_i, D_i is the relative deadline and T_i the (minimum) distance between consecutive job releases in the periodic or sporadic model respectively. A task τ_i may release a potentially infinite sequence of jobs, each with release time $r_i^q \geqslant r_i^{q-1} + T_i$ and with absolute deadline $d_i^q = r_i^q + D_i$. We concentrate on task-sets where for all the tasks $D_i \leqslant T_i$, which is commonly termed the constrained deadline task model. The base scheduling policy considered is global fixed task priority scheduling. Each task has a priority associated with it. Correspondingly every job from a given task τ_i executes at task τ_i priority. For each task τ_i, $\mathrm{lp}(i)$ denotes the set of tasks with lower priority than τ_i, similarly $hp(i)$ stands for the set of higher priority tasks and $hep(i)$ denotes the set of tasks of equal or higher priority than τ_i. Tasks are assumed to be composed of multiple fixed non-pre-emptive regions. A fixed non-pre-emptive region is defined as some workload which when executed does not suffer any interference from higher priority workload. In practice this may be accomplished by placing explicit pre-emption points at the boundaries of these regions such that if one boundary is crossed τ_i can only be pre-empted once the next boundary is reached. The worst-case execution time distance between the boundaries of the region is the length of the non-pre-emptive region. For each task the maximum length of such a non-pre-emptive region is denoted by Q_i. It is assumed that the last non-pre-emptive region of any task is not smaller than any other non-pre-emptive region preceding it. The platform considered is composed of m unit-speed identical processors denoted as $\pi_1, \pi_2, \cdots, \pi_m$ (all processors have the exact same computing capabilities). Each job can only execute on a single core at any point in time (i.e. jobs cannot execute workload in parallel). The migration and pre-emption delays are assumed to be negligible.

III. Related Work

An upper-bound on the interference a given job suffers when scheduled on a multiprocessor concurrently with other higher priority workload was first proposed by Baker [6]. This technique was later reused by Bertogna et al. in order to devise a response time analysis for global fixed task priority and global EDF [7]. The response time analysis for global EDF was then refined by Baruah [8] where the worst-case response time of a task is computed in a busy period starting with $m-1$ tasks with carry-in[1] and the remainder having a synchronous release in the beginning of the same time interval. Later Guan [9] adapted the approach of Baruah to global fixed task priority and proved that an upper-bound on the worst-case response time for this scheduling policy can be constructed by considering $m-1$ higher priority tasks with carry-in in the beginning of a time interval and a synchronous release of the remaining higher priority tasks with the task under analysis. This result was later generalised by Davis and Burns [10] who showed that a scenario with a maximum of m-1 tasks with carry-in is the worst-case not only for the sufficient test proposed by Guan, but also in general, i.e. for any exact test.

[1]A task is referred to as having carry-in if at the start of the interval considered in the analysis, it has a job that has been released but not yet completed.

The limited pre-emptive scheduling literature is prolific in what concerns single core scheduling. Restricting pre-emption points presents a viable way to address the problem of pre-emption delay. The mechanism of limited pre-emption termed *fixed non-pre-emptive region scheduling* was first proposed by Burns et al. [11] for single processors. A more complete schedulability analysis of the limited pre-emptive model was presented by Bril in [12], [13] where it is shown that more than one frame in a given priority level busy period has to be tested for the temporal correctness.

A similar mechanism termed *floating non-pre-emptive region scheduling* was proposed by Baruah [14] for single processor EDF. In a *floating non-pre-emptive region* the positions of the non-pre-emptive region in the task execution are variable. This contrasts with the *fixed non-pre-emptive region* model which statically defines the non-pre-emptive regions. The *floating non-pre-emptive region* model was later adapted by Gang Yao et al. [15] for single processor fixed task priority scheduling where a bound on the length of the floating non-pre-emptive regions for each task is provided. Gang Yao et al. presented another methodology to compute the maximum length of the fixed non-pre-emptive regions for single processor fixed task priority scheduling [16]. In this situation the computed length of the fixed non-pre-emptive regions are generally larger than in previous work, as the last chunk of a task's execution is not subject to further pre-emptions. This enables a schedulability increase in comparison to fully pre-emptive disciplines and a further reduction in the number of pre-emptions observed in the schedule. Gang Yao et al. provide a comparison of all available methods described so far in the literature [17] regarding restricted pre-emptive scheduling using single processor fixed task priority. In this paper we revisit the limited pre-emptive model and devise the theory for multiprocessor global fixed task priority scheduling focusing solely on the *fixed non-pre-emptive region* model.

The pre-emption delay estimation problem using the *fixed non-pre-emptive region* model in single processors was presented by Bertogna et al. [18]. In order to reduce cache related pre-emption delay (CRPD), the use of fixed non-pre-emptive areas of code is exploited. In this way the maximum CRPD is decreased and overall system response time is enhanced.

The limited pre-emptive models have yet to be fully addressed in the multiprocessor domain. Nevertheless some works employing more restrictive solutions exist. Guan et al. presented a schedulability analysis for global non-pre-emptive fixed task priority scheduling [19]. In this work all the tasks are deemed non-pre-emptive, which has the implication that when jobs are dispatched onto a processor they execute until completion. A less restrictive policy is presented by Lee and Shin [20] where tasks can be either fully pre-emptive on fully non-pre-emptive. This work was devised for global EDF. Davis et al. [21] adapted previous work [22] on optimal fixed priority scheduling with deferred pre-emption for single processor systems to the multiprocessor case. However, in the case of multiprocessor systems, the model was restricted to tasks having only a single, final non-pre-emptive region at the end of their execution.

In this paper we present an even less restrictive model where tasks comprise a collection of non-pre-emptive regions of execution. This model is then a generalization of [19], [20] and [21].

IV. Global Fixed Task Priority response time analysis

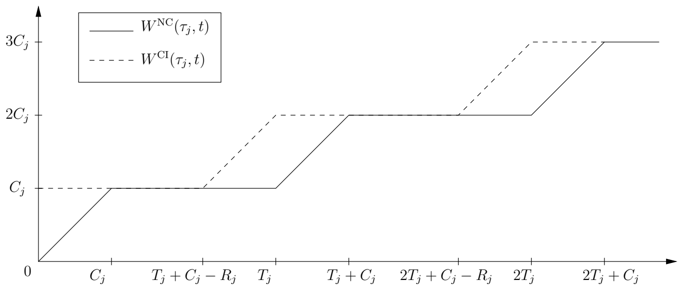


Fig. 1: Functions $W^{\mathrm{NC}}(\tau_j,t)$ and $W^{\mathrm{CI}}(\tau_j,t)$ depiction for a given task τ_j

In global fully pre-emptive fixed priority scheduling, if a task τ_j has no pending workload at the beginning of an interval, an upper bound on the workload which it may execute in an interval of length t is given by [9]:

$$W_j^{\mathrm{NC}}(t) = \left\lfloor \frac{t}{T_j} \right\rfloor \times C_j + \min(t \mod T_j, C_j) \tag{1}$$

In (1) a release of a job from task τ_j is assumed to occur at time instant 0 and subsequent releases happen at the minimum inter-arrival time, in fact leading to a worst-case amount of workload released in an interval of length t when no carry-in is present.

Similarly, if the same task τ_j might have some pending workload released before the beginning of the interval, a conservative upper-bound on the workload it may execute in an interval of length t is given by [9]:

$$W_j^{\mathrm{CI}}(t) = \left\lfloor \frac{\max(t-C_j,0)}{T_j} \right\rfloor \times C_j + C_j$$
$$+ \min\left([[t-C_j]_0 \mod T_j - (T_j - R_j)]_0, C_j\right) \tag{2}$$

In (2) a job of task τ_j is assumed to be released before the start of the interval and that it will execute the bulk of the workload after the interval starts. The upper-bound present in (2) assumes that the carry-in job from task τ_j was subject, since its release at time instant $-R_j^{UB} - C_j$ until the beginning of the interval, to the worst-case scenario such that this job has not executed any workload yet. The execution of the workload will terminate R_j time units after the job was released. Subsequent jobs are released with the minimum inter-arrival separation. The second job is then released at time instant $T_j - R_j + C_j$. This constitutes a conservative estimation of the workload a task τ_j executes in a given time interval of length t if there exists some pending workload released at some point before the beginning of the interval.

For the same task τ_j (2) is an upper-bound on (1), this can be observed from the graphical representation of both functions presented in Figure 1. The difference between the upper-bound considering a carry-in scenario and one where no carry-in is considered yields:

$$W_j^{\mathrm{diff}}(t) = W_j^{\mathrm{CI}}(t) - W_j^{\mathrm{NC}}(t) \tag{3}$$

In [9] it is shown that a conservative upper-bound on the amount of higher priority workload which will execute during the response time of a job from task τ_i is constructed by assuming that some $m-1$ higher priority tasks have pending workload at the time of τ_i's job release and that a synchronous release of jobs from the remaining higher priority tasks occurs at the same instant as τ_i's job release. Writing this upper-bound in more formal terms yields:

$$\Omega_i(t) \overset{\mathrm{def}}{=} \left(\sum_{l=1}^{m-1} \max_{\tau_j \in \mathrm{hp(i)}}^{\ell} W_j^{\mathrm{diff}}(t) \right) \tag{4}$$
$$+ \sum_{\tau_j \in \mathrm{hp(i)}} W_j^{\mathrm{NC}}(t) \tag{5}$$

Where $\max^{\ell}_{\tau_h \in \tau_1, \cdots, \tau_{i-1}}$ returns the ℓ^{th} greatest function value along the higher priority task's workload dimension. In a situation where no higher priority task has carry-in workload at the beginning of the time interval, a sum over all the $W^{\mathrm{NC}}(\tau_j,t)$ functions of tasks of higher priority than τ_i would yield an upper-bound on the higher priority workload that would execute in the time interval of length t. Since we know that in the worst-case some $m-1$ tasks present carry-in workload at the beginning of the interval, then this additional workload will never exceed the difference between the maximum workload in a no carry-in situation and the carry-in situation. By choosing the $m-1$ higher priority tasks which for a given interval length display the biggest difference between the no carry-in and carry-in situation and by summing these differences over all functions of no carry-in, we have an upper-bound on the workload which higher priority tasks may execute in the interval of length t.

In [6] Baker showed that in a multi-core platform composed of m identical cores, a unit of higher priority workload can interfere with the execution of task τ_i by at most $\frac{1}{m}$ time units. Consider that some task τ_j is executing on one processor during one time unit. By executing on this processor it will not prevent τ_i from getting hold of some other processing entity as there exist $m-1$ others in the platform. In order for τ_i to be prevented from executing, the m processors need to be busy. In order for m cores to be busy for one time unit then m time units of higher priority workload need to execute on the overall platform. Combining (5) and this fact enables the statement of the upper-bound on the overall interference a task τ_i is subject to in a time interval of length t. This is written as:

$$I_i(t) = \frac{\Omega_i(t)}{m} \tag{6}$$

When considering the fully pre-emptive global fixed task priority scheduling policy, by exploiting the upper-bound on the interference by higher priority workload, the following sufficient schedulability test is devised:

$$\forall \tau_i \in \mathcal{T}, \exists t \in [0, D_i] : I_i(t) + C_i \leqslant t \tag{7}$$

A task-set is said to be schedulable if for all task $\tau_i \in \mathcal{T}$ the condition in (7) holds. The schedulability test presented in (7) is a sufficient test. This means that some task-sets may indeed manage to meet all the deadlines even if for some tasks the condition in (7) is not met. Nevertheless if the condition is met for all the tasks in the taskset then the timing requirements of all the tasks are guaranteed to be met at run-time.

Similarly to the single processor case this condition need not be tested for all of the values in the continuous interval

$[0, D_i]$ but rather for a finite number of time instants. Both functions $W_j^{\mathrm{NC}}(t)$ and $W_j^{\mathrm{CI}}(t)$ are piecewise linear functions, i.e. they may be defined as a set of linear functions in distinct time intervals. As a consequence of this, the $\Omega_i(t)$ function itself may be defined as a set of linear functions in distinct time intervals as well. Hence the relation $t - I_i(t)$ is maximized in the $[0, D_i]$ interval for some value $t \in \Gamma_i$. The set Γ_i encloses the Γ_i' set of points where the first derivative of the function $\Omega_i(t)$ changes in value and the time instant of the deadline D_i of τ_i. The $\Gamma_i' \cup \{D_i\}$ set of points contains the points of interest when trying to maximize the value of $t - I_i(t)$.

For brevity and space constraints the derivation of the set of points Γ_i is ommited from this document. The full details on its derivation are provided in a techical report [23].

When the schedulability condition is tested over a discrete set of points the response time computation can still be efficiently carried out:

$$t_1 = \min\{t \in \Gamma_i | I_i(t) + C_i \leqslant t\} \tag{8}$$

$$t_2 = \max\{t \in \Gamma_i | t < t_1\} \tag{9}$$

$$R_i^{UB} = t_1 + \frac{I_i(t_1) + C_i - t_1}{1 - \frac{I_i(t_2) - I_i(t_1)}{t_2 - t_1}} \tag{10}$$

The quantity $R_i^{UB} \in (t_2, t_1]$ is the intersection between the line segment

$$I(t) = \frac{I(t_1) - I(t2)}{t_1 - t_2} \times t + I(t_1) - \frac{I(t_1) - I(t2)}{t_1 - t_2} \times t_1 \tag{11}$$

defined $\forall t \in (t_2, t_1]$ and the supply line $f(t) = t$.

The formulation of the sufficient schedulability condition presented in this work is equivalent to the one in [9]. We chose to formulate it not as a fixed point algorithm but rather as a condition over an interval in order to ease the definition of the parameters that we compute further on in this work.

V. Limited Pre-emptive Scheduling Policies

In a multi-core platform the stock global fixed priority fully pre-emptive scheduling discipline may be informally described as a policy where at any time t the m highest priority tasks with available workload execute on the m processors comprising the platform.

When tasks are composed of both pre-emptible and non-pre-emptible workload then more complex protocols may be devised reflecting the more complex nature of the workload. In this work two scheduling policies are considered:

1) *Regular Deferred Scheduling* (RDS): At any point in time, the pre-emptible jobs executing in any processor are eligible for being pre-empted by a higher priority job;
2) *Adapted Deferred Scheduling* (ADS): At any point in time a pre-emption can only occur if the lowest priority running job is pre-emptible, in which case the lowest priority running job is pre-empted from the processor on which it runs and the highest priority waiting job is dispatched onto the same processor.

Bear in mind that these are only two generalisations of the regular fully pre-emptive scheduling discipline. The RDS policy is the straightforward derivation of the fully pre-emptive scheduler whereas the ADS is an adaptation which enables some interesting properties with respect to lower priority interference to be achieved.

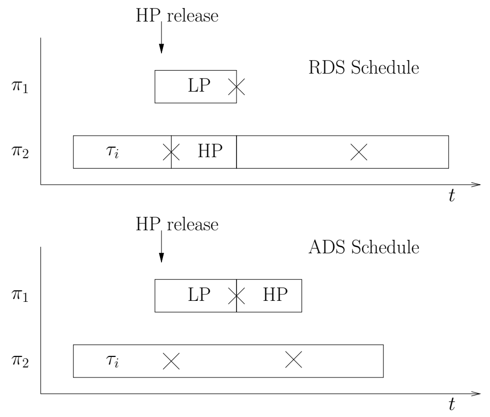


Fig. 2: Possible Priority Inversion After a Job From τ_i Commences Execution in RDS

In the RDS scheduling policy, a higher priority job from τ_i might suffer interference from lower priority non-pre-emptive regions more than once after τ_i has commenced execution. A situation where said priority inversion occurs after the start of τ_i execution may be observed in the top schedule displayed in Figure 2. In Figure 2 the crosses represent fixed pre-emption points. The bottom schedule in the same picture displays the ADS schedule for the specific workload pattern. In this case, before τ_i is pre-empted, all other lower priority workload has to be pre-empted from the platform. A task can only be pre-empted if it is the task currently running upon some processor such that it has the lowest priority among all tasks currently executing in the system.

Once a task starts to execute its last non-pre-emptive region it ceases to suffer interference, as a consequence it is only subject to interference during the execution of the first $C_i - Q_i$ units of workload. The schedulability test then becomes:

$$\exists t \in [0, D_i - Q_i] :$$
$$t - \frac{1}{m} \times \left(WA_i^{diff}(t) + W_i^{NC}(t) + A_i^{NC}(t)\right) - (C_i - Q_i) > 0 \tag{12}$$

The corresponding upper-bound on the response time of a given task τ_i can thus be computed as:

$$R_i^{UB} = \min\{t | t - \frac{1}{m} \times \left(WA_i^{diff}(t) + W_i^{NC}(t) + A_i^{NC}(t)\right) - (C_i - Q_i) > 0\} + Q_i \tag{13}$$

Similarly to the fully pre-emptive scenario, the term $W_i^{NC}(t)$ upper-bounds the maximum interference that higher priority workload may induce on τ_i when no higher priority carry-in exists. The $A_i^{NC}(t)$ function characterizes the maximum amount of interference due to lower priority workload released inside the interval of interest, which may exhibit non-pre-emptive regions and hence prevent higher priority workload from executing on the processors. The function $WA_i^{diff}(t)$ encapsulates the maximum interference contribution from carry-in workload. This is workload which was released before or immediately before the beginning of the interval of interest and which will be executed inside the interval of interest. As in previous works on fully pre-emptive scheduling, the worst-case scenario with respect to higher priority workload occurs when there are at most $m-1$ higher priority tasks with carry in jobs [9], [10]. In the case of non-pre-emptive regions there might exist at most m lower priority tasks which were executing immediately before the start of the interval of interest. If some core is executing lower priority workload then this implies that this core cannot be executing higher priority carry-in in the beginning of the interval of interest. As a consequence then, if there are k lower priority tasks executing non-pre-emptively in the beginning of the interval of interest, then there can be at most $m-k$ higher priority tasks with carry-in. The maximum additional workload due to the k lower or equal priority tasks executing in the beginning of the interval of interest is denoted by A_i^k. The computation of an upper-bound of A_i^k given a set of lower or equal priority non-pre-emptive regions is the subject of the subsequent sections.

The $WA_i^{diff}(t)$ function is defined as follows:

$$WA_i^{diff}(t) \overset{\text{def}}{=} \max_{k\in\{1,\cdots,m\}} \left\{ A_i^k + \sum_{l=1}^{m-k} \max_{\tau_h\in\tau_1,\cdots,\tau_{i-1}}^{\ell} W^{\text{diff}}(\tau_h,t) \right\} \quad (14)$$

In the ADS policy $A_i^{NC}(t)=0$. Lower priority tasks may execute in other processors while τ_i is executing, but higher priority workload will only be able to pre-empt τ_i when τ_i is the lowest priority task executing on any processor. As a consequence of this, lower priority workload can only interfere with τ_i execution if they are currently executing once τ_i is released. We note that this is the key difference between ADS and regular fixed priority scheduling with deferred pre-emption (RDS).

Contrary to the single processor limited pre-emptive theory, where multiple jobs from τ_i in a $level{-}i$ busy period need to be checked for the temporal correctness, this is not the case in the presented schedulability condition ((12)). As a consequence, when $m=1$ the analysis provided is still safe albeit pessimistic as the provided test is sufficient but not necessary, whereas the test available in the literature for single processor is both necessary and sufficient [12], [13].

Theorem 1 (Correctness of the Schedulability Condition (12))**.** *Let us assume that the term $A_i^k(t)$, in* (14) *, gives an upper-bound on the workload generated by all the jobs from tasks with priority lower than or equal to i, released before time t and executed non-pre-emptively on k processors (we will show later how to compute this upper-bounds). If* (14) *is satisfied for all task τ_i then the task-set is schedulable.*

Proof: For a given $k \in [1,m]$, the equation in the brackets of (14) gives an upper-bound on the carry-in workload at time t, where (i) k processors execute non-pre-emptive workload coming from k lower or equal priority jobs released before time t, and (ii) $m-k$ processors execute pre-emptive workload coming from $m-k$ higher priority jobs released before time t.

Therefore, $WA_i^{diff}(t)$ as defined in Equation (14), which takes the max for all k, is an upper-bound on the carry-in workload at time t. From the definitions of $W_i^{NC}(t)$ and since $A_i^{NC}(t)=0$, it holds for a given time-instant t that the sum $WA_i^{DIFF}(t)+W_i^{NC}(t)$ gives an upper-bound on the total workload at time t (including both carry-in and non carry-in workload from lower- and higher-priority tasks). The correctness of the condition given by Inequality (12) immediately follows from the meaning of this sum, i.e., if the condition is satisfied for a given t, then it means that any job from task τ_i will always be able to execute for at least $C_i - Q_i + \epsilon$ time units within $D_i - Q_i$ time units from its release. Given that every job of τ_i will get the highest priority after executing for $C_i - Q_i + \epsilon$ time units, it implies that all jobs of τ_i will have to execute its (at most) Q_i remaining time units within the last Q_i time units to its deadline, which it will always do. ■

In order to assess the schedulability of τ_i it is important to quantify A_i^k where $k \in \{1,\cdots,m\}$. The derivation of upper-bounds for A_i^k is the subject of the subsequent section.

An alternative way of writing the schedulability condition for the ADS policy is:

$$\beta_i^k(Q_i) \overset{\text{def}}{=} \max_{t\in[0,D_i-Q_i]} \Big\{ m\times t - W_i^{NC}(t) - m\times(C_i-Q_i) - \sum_{l=1}^{m-k} \max_{\tau_h\in\tau_1,\cdots,\tau_{i-1}}^{\ell} W^{\text{diff}}(\tau_h,t) \Big\} \quad (15)$$

The task-set is deemed schedulable for a given set of last non-pre-emptive region for all the task $\tau_i \in \mathcal{T}$ if:

$$\forall \tau_i \in \mathcal{T}, \forall k \in \{1,\cdots,m\}, \beta_i^k(Q_i) \geqslant A_i^k \quad (16)$$

The computation of the Function (15) can be performed by analyzing the limited set of time instants Γ_i described previously.

$$F_i^k(t) = \max_{\{t'\in\Gamma_i\cup\{t\}|t'\leqslant t\}} \Big\{ m\times t' - W_i^{NC}(t') - m\times C_i - \sum_{l=1}^{m-k} \max_{\tau_h\in\tau_1,\cdots,\tau_{i-1}}^{\ell} W^{\text{diff}}(\tau_h,t') \Big\} \quad (17)$$

Equation (15) may then be rewritten as:

$$\beta_i^k(Q_i) = F_i^k(D_i - Q_i) + Q_i \times m \quad (18)$$

VI. Maximum Interference from Lower or Equal Priority Non-Pre-emptive Regions in ADS

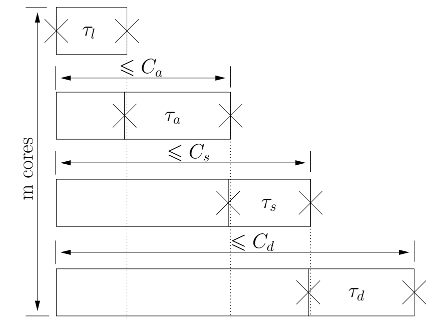


Fig. 3: Maximum Interference Function Due to m Non-pre-emptive Regions of Lower or Equal Priority Tasks

The ADS scheduling policy considered in this work dictates that a task can only be dispatched to run on one of the m cores, at a time instant t, when either a core is idle or a pre-emption point of the lowest priority task running on any m processors has been reached. The task which is pre-empted is then the lowest priority task running on any of the cores at time t.

The worst-case interference pattern generated by the lower or equal priority non-pre-emptive region execution is represented in Figure 3 for a platform where $m = 4$. In Figure 3 the crosses represent fixed pre-emption points. In a scenario where m processors are executing lower or equal priority workload and several higher priority releases occur, the first task to be pre-empted is τ_l which is of lower priority than the remainder ($\tau_l \prec \tau_a \prec \tau_s \prec \tau_d$) at time t_l when a pre-emption point from τ_l is reached. In the worst case scenario task τ_a enters a non-pre-emptive region at time $t_l - \epsilon$ and its next pre-emption point is reached at $t_l + \epsilon + Q_a$. Subsequently in the worst-case situation τ_s has entered a non-pre-emptive region just before the pre-emption point of τ_a was reached.

Let us assume the total ordered set $\mathcal{LQ}_i = \{Q_n, \cdots, Q_i\}$ where, if $d > l$ then $Q_d \succ Q_l$. Each element of the $\mathcal{LQ}_i$ set represents the length of the non-pre-emptive region of task with priority equal or lower than τ_i, the priority ordering among tasks is represented by the total ordering of the elements of the set.

Given a subset $\mathcal{SQ}_i$ of $\mathcal{LQ}_i$ with k elements one can compute the A_i^k area accurately: Algorithm 1 places non-pre-emptive regions in $\mathcal{SQ}_i$ in priority order. The first element Q_1 is the lowest priority element in $\mathcal{SQ}_i$ (observe that in case the subscript ℓ in Q_ℓ references the ℓth element in the totally ordered set $\mathcal{SQ}_i$), in the worst case its pre-emption point will be reached at $t_1 = Q_1$. At the end of the first iteration the variable $span$ is equal to Q_1. The span variable keeps track of the rightmost pre-emption point from all the tasks when these are placed in priority ordering in order to construct the maximum A_i^k from the set of $\mathcal{SQ}_i$ values. Some task τ_s may not have enough C_s such that its pre-emption point would be placed at $span + Q_s$, in which case the $span$ variable either remains constant if $C_s < span$, or $span = C_s$ otherwise.

Algorithm 1: Low Priority Interference Computation from a $\mathcal{SQ}_i$ subset of k Lower or Equal Priority Non-pre-emptive Regions

```
Input  : $\mathcal{SQ}_i, i, k$
Output: $A_i^k$
$A = 0$
span $= 0$
for $y \in \{k, \cdots, 1\}$ do
    if $C_y \leqslant span + Q_y$ then
        if $C_y > span$ then
            $span = C_y$
        $A = A + C_y$
    else
        $A = A + span + Q_y$
        $span = span + Q_y$
return $A_i^k$
```

Algorithm 1 takes as input a set of k tasks with lower or equal priority than τ_i and returns the *exact* worst-case interference from these k tasks on task τ_i – as a result, Algorithm 1 can be used to derive the exact worst-case interference from the lower priority tasks on any task τ_i. However, doing so would require to enumerate all possible subsets of k tasks out of the set of all the tasks with a lower or equal priority than τ_i and the computation of the exact interference would be of exponential complexity.

As a compromise between accuracy and computation time, we propose below three methods that derive an *upper-bound* on the worst-case lower priority interference. The first one is the simplest (the least accurate and fastest) as it factors neither the task priorities, nor the WCET constraints in the computation and considers the maximum non-pre-emptive region length from all lower priority non-pre-emptive regions.

ADS Blocking Estimation 1.

The most straightforward method relies on considering the largest lower priority non-pre-emptive region and constructing the A_i^k area with it in conjunction with at most a single instance of the non-preemptive region of task τ_i so as to encompass the self-pushing effect. This bound is stated in (19).

$$A_i^k \leqslant \frac{k-1}{2} \times (k+2) \times \max\{\mathcal{LQ}_i \setminus \{Q_i\}\} + \max\{\mathcal{LQ}_i\} \tag{19}$$

ADS Blocking Estimation 2.

The second method is slightly more complex, it considers a variety of last non-pre-emptive regions present in $\mathcal{SQ}_i$. The largest interference is obtained when the largest element of $\mathcal{LQ}_i$ is accounted for k times (assuming the lowest priority for this task), the second largest is added up $k-1$ times (assuming the second lowest priority for this task), etc., until the k^{th} largest element which is only considered once. This upper-bound is stated in (20).

$$A_i^k \leqslant \sum_{j=1}^{k} Q_j^{\max} \times (k-j+1) \tag{20}$$

where $Q_j^{\max}$ denotes the j^{th} largest element in the set $\mathcal{LQ}_i$.

ADS Blocking Estimation 3.

By taking the priority ordering among the lower or equal priority tasks into account, it is possible to construct a less pessimistic upper-bound on the A_i^k quantity. The problem can be formulated as follows: Find k $\mathcal{LQ}_i$ element indexes $\{x_1, x_2, \ldots, x_k\}$ such that for all $j \in [1, k]$: $x_j \in [1, n-i]$, $\tau_{x_j} \prec \tau_{x_{j+1}}$ and

$$\sum_{j=1}^{k} Q_{x_j} \times (k-j+1) \tag{21}$$

is maximum. The third method that we propose solves this problem. First, let us reformulate it as follows.

Problem 1: Given a set of non-negative values $\{Q_1, Q_2, \ldots, Q_{n-i+1}\}$ ordered by task priority (note that these subscripts relate to the position of the element in the totally ordered set, higher priority is associated to higher set index value) and a non-negative integer $k \leqslant n-i+1$, we construct a table T of k rows and $n-i+1$ columns such that the value $v_{y,z}$ of the cell in row y and column z is set to $v_{y,z} = (k - y + 1) \times Q_z$ (rows are indexed from 1 to k and columns from 1 to $n - i + 1$). The problem consists of finding S for which there exists $\{x_1, x_2, \ldots, x_k\}$ such that each x_j, $1 \leq x_j \leq n - i + 1$, denotes the index of a column and it holds that

1) $x_j < x_k$, $\forall\ 1 \leq j < k$, and
2) $S = \sum_{y=1}^{k} v_{y,x_j}$ is maximum

It is easy to see that a solution S to problem 1 is also a maximum value for the sum of (21), and thus an upper-bound on A_i^k.

Algorithm 2: Algorithm to Compute S

```
The idea is to construct another table $T'$ based on $T$
as follows:
    1) As $T$, the table $T'$ has $k$ rows and $n - i + 1$
       columns.
    2) We set $v'_{1,1} = v_{1,1}$
    3) For the other cells $v'_{1,z}$ of the first row, with
       $z = 2, \ldots, n - i + 1$, we set
       $v'_{1,z} = \max(v_{1,z}, v'_{1,z-1})$
    4) For each row $y = 2, \ldots, k$:
        a) We set $v'_{y,y} = v_{y,y} + v'_{y-1,y-1}$
        b) The other cells $v'_{y,z}$ of the $y^{\text{th}}$ row, with
           $z = y + 1, \ldots, n - i - k + y + 1$, we set
           $v'_{y,z} = \max(v_{y,z} + v'_{y-1,z-1}, v'_{y,z-1})$.
Finally, we have $S = v'_{k,n-i+1}$
```

Lemma 1. *$S = v'_{k,n-i+1}$ is a solution to problem 1.*

Proof: The proof is obtained by (double) induction, first on y (row index) and then on z (column index). We show for all y and z, with $1 \leq y \leq k$ and $1 \leq z \leq n - i + 1$, that $S = v'_{y,z}$ is the maximum value of the sum $\sum_{\ell=1}^{y} v_{\ell,x_\ell}$, assuming that the k variables $x_1, x_2, \ldots x_k$ are such that $x_\ell \in [1, z]$, $\forall \ell$.

base case: $y = 1$ and $z = 1$.

The case is straightforward: $v'_{1,1} = v_{1,1}$ and thus $S = v'_{1,1}$ is the maximum value of the sum $\sum_{\ell=1}^{y} v_{\ell,x_\ell}$ where there is only a single variable x_1 and $x_1 = 1$ is the only choice.

Inductive step on z: $y = 1$ and $1 < z \leq n - i + 1$.

By the induction, we assume that for $y = 1$ and for all $p \in [1, z - 1]$, $S = v'_{1,p}$ is the maximum value of the sum $\sum_{\ell=1}^{y} v_{\ell,x_\ell}$ where there is a single variable x_1 and x_1 is chosen within $[1, p]$.

By construction, for all $z = 2, \ldots, n - i + 1$, the value $v'_{1,z}$ is defined as $v'_{1,z} = \max(v_{1,z}, v'_{1,z-1})$ and thus either $S = v'_{1,z}$ is equal to $v'_{1,z-1}$ (the maximum previously recorded and x_1 is chosen within $[1, z - 1]$) or it is equal to $v_{1,z}$, in which case $S = v_{1,z}$ is the maximum and $x_1 = z$ leads to the maximum sum $\sum_{\ell=1}^{1} v_{\ell,x_\ell}$ $(= S)$.

Inductive step on y, base case on z: $1 < y \leq k$ and $z = y$.

The value $v'_{\ell,\ell}$ is defined as $v'_{z,z} = v_{z,z} + v'_{z-1,z-1} = \sum_{\ell=1}^{z} v_{z,z}$. As we must have $x_\ell < x_{\ell+1}$ for all $1 \leq \ell < y$, the only choice for the y variables $x_1, x_2, \ldots x_y$ is to have $x_\ell = \ell$ for all $\ell \in [1, y]$. Therefore, $S = v'_{z,z}$ is the maximum value of the sum $\sum_{\ell=1}^{z} v_{\ell,\ell}$.

Inductive step on y and z: $1 < y \leq k$ and $1 < z \leq n - i + 1$.

By the induction, we assume that for all $r \in [1, y - 1]$ and for all $p \in [1, z - 1]$, $S = v'_{r,p}$ is the maximum value of the $\sum_{\ell=1}^{r} v_{\ell,x_\ell}$ where the variables $x_1, x_2, \ldots, x_r$ are chosen within $[1, p]$.

By definition, we know that the maximum value of the sum $\sum_{\ell=1}^{y} v_{\ell,x_\ell}$ assuming that the y variables $x_1, x_2, \ldots, x_y$ are chosen within $[1, z]$, is equal to the maximum between

1) the maximum value of the sum $\sum_{\ell=1}^{y} v_{\ell,x_\ell}$ assuming that the y variables $x_1, x_2, \ldots, x_y$ are chosen within $[1, z - 1]$, and
2) the maximum value of the sum $\sum_{\ell=1}^{y-1} v_{\ell,x_\ell}$ assuming that the $y - 1$ variables $x_1, x_2, \ldots, x_{y-1}$ are chosen within $[1, z - 1]$ and $x_y = z$.

This is reflected at step (4b) where $v'_{y,z}$ is set to the maximum of both.

It is thus true that $v'_{y,z}$ holds the maximum value of the sum $\sum_{\ell=1}^{y} v_{\ell,x_\ell}$ assuming that the variables $x_1, x_2, \ldots, x_y$ are chosen within $[1, z]$. As the result holds for $y = k$ and $z = n - i + 1$, we have that $S = v'_{k,n-i+1}$ is a solution to problem 1. ■

Algorithm 2 tests at most $(n-i-k+1) \cdot k$ different scenarios since it traverses at most $n - i - k + 1$ elements k times, which compares favorably with the brute force approach which would test the $\binom{n}{k}$ different legal scenarios.

So far the A_i^k area has been upper-bounded by only taking into consideration the priority ordering between the non-pre-emptive regions of lower or equal priority tasks. It might be the case in fact that the worst-case execution time of the lower or equal priority tasks does not enable the result obtained with Algorithm 2 ever to occur in practice.

In the worst-case the blocking area cannot exceed the sum of the k largest lower-priority task WCET:

$$A_i^k \leqslant \sum_{j=1}^{k} \max_{\ell \in lp(i)}^{j} C_\ell \tag{22}$$

VII. Enhancing System Predictability with Fixed Non-Pre-emptive Regions

Let us consider a scenario where the priority ordering among tasks is provided *a-priori*. We intend to compute a set of non-pre-emptive regions for each task such that the number of pre-emptions observed in a schedule is reduced. A first approach to solving this problem is presented in Algorithm 3. The task-set is parsed starting from the lowest priority task τ_n. At each priority level i, the set of minimum Q_i^k values which render the m schedulability constraints (16) are found. From these m values the largest one is chosen. Since the $\beta_i^k(Q_i)$ functions are monotonically non-decreasing, if $Q_i^{k'} > Q_i^k$ and $\beta_i^k(Q_i^k) = A_i^k$ then $\beta_i^k(Q_i^{k'}) \geqslant A_i^k$. Hence choosing the maximum Q_i^k out of all the m minimum values which make the m inequalities true will still ensure the attainment of the schedulability condition. If, for any of the m schedulability conditions there exists no Q_i quantity for which $\beta_i^k(Q_i) = A_i^k$ then the task-set is deemed unschedulable.

Algorithm 3: Minimum Last Non-pre-emptive Region Length (Q_i) Assignment

```
for $i \in \{n, \cdots, 1\}$ do
    for $k \in \{1, \cdots, m\}$ do
        if $\exists \{Q_i | \beta_i^k(Q_i) = A_i^k\}$ then
            $Q_i^k = \{Q_i | \beta_i^k(Q_i) = A_i^k\}$
        else
            return UNSCHED
    $Q_i = \max_{1 \leqslant k \leqslant m}\{Q_i^k\}$
return SCHED
```

Lemma 2 (Minimum Non-pre-emptive Region Assignment)**.** *Algorithm 3 provides the minimum set of Q_i values $\forall \tau_i \in \mathcal{T}$ such that the task-set is schedulable under ADS with a given priority assignment*

Proof: Proof by induction. For task τ_n the quantity Q_n computed by Algorithm 3 is the smallest last non-pre-emptive region length such that task τ_n is schedulable. As a consequence, the blocking that τ_n induces on the higher priority task is the minimum possible such that τ_n is schedulable.

Inductive step: Algorithm 3 yields the minimum last non-pre-emptive region length for a task $\tau_i, 1 \leqslant i \leqslant n$ such that τ_i is schedulable. As a consequence of this the set of task $\{\tau_i, \cdots, \tau_n\}$ induces the lowest possible worst-case blocking to the higher priority workload such that those tasks are schedulable.

If for the same priority assignment any value $Q'_i \in \{Q'_i, \cdots, Q'_n\}$ it would happen that $Q'_i < Q_i$, then task τ_i would be unschedulable as a consequence. ■

Theorem 2. *The ADS policy dominates the fully pre-emptive fully pre-emptive and fully non-pre-emptive global fixed task priority with respect to schedulability*

Proof: This result is easily proven by observing that according to Lemma 2 the Q vector outcome of Algorithm 3 is the smallest such that the taskset is schedulable. As a consequence, if $\mathcal{T}$ is schedulable under fully pre-emptive global fixed task priority the set Q resulting from Algorithm 3 is such that $\forall Q_i \in \{Q_1, \cdots, Q_n\} : Q_i = 0$. Otherwise if $\mathcal{T}$ is not schedulable with fully pre-emptive global fixed task priority but it is with ADS then $\exists Q_i \in Q : Q_i > 0$. Similarly if a task is only schedulable with fully non-pre-emptive the Q vector produced by algorithm 3 would be such that each $Q_i = C_i$. Since $A_i^k \leqslant \sum_{j=1}^{k} \max^{j}_{\ell \in lp(i)} C_l$ the maximum blocking lower priority tasks induce in ADS can in the worst-case be equal to that of fully non-pre-emptive and never greater. In a situation where $\forall i, Q_i = C_i$ the ADS policy is equivalent to the fully non-pre-emptive policy (i.e. the schedules produced are identical). ■

Having the mechanism to produce the minimum set of Q values which ensures the schedulability of the task-set in ADS we intend to compute a set of non-preemptive regions where at least some of its constituents are larger than the corresponding components of the minimum vector but never smaller. Having larger Q_i potentially leads to a smaller number of preemptions in the actual schedule as will be showcased in the Experimental Section.

Algorithm 4: Last Non-pre-emptive Region Length (Q_i) Assignment

```
for $i \in \{1, \cdots, n\}$ do
    for $k \in \{1, \cdots, m\}$ do
        $Q_i^k = \max\{Q | \forall j \in hep(i), \beta_j^k(Q_i) \geqslant A_j^k\}$
    $Q_i = \min_{1 \leqslant k \leqslant m}\{Q_i^k\}$
```

Algorithm 4 takes as input the minimum Q vector ensuring schedulability. Contrary to the minimum Q vector computation procedure we now take a top down approach (i.e. starting from the highest priority to the lowest). The resulting Q' vector has all its elements larger or equal to the minimum Q_i vector since by definition this is the smallest possible ensuring schedulability. At each priority level the maximum Q_i quantity is assigned which still preserves the schedulability of higher priority tasks. It is considered that any remainder lower or equal priority task τ_ℓ has a Q_ℓ equal to the maximum between any Q_j where $j \in hep(i)$ and the minimum Q_ℓ which renders τ_ℓ schedulable. This is due to the plausible scenario where a tasks with lower priority than τ_i processed in further iterations requesting a greater value than its minimum Q_ℓ. Since at the given iteration we are unaware of future developments and in order to reduce complexity the future requests of lower priority tasks are limited to the values known to us in the current iteration. These are the set of minimum last non-pre-emptive region lengths ensuring the schedulability of each lower priority task and the set of assigned higher priority task last non-pre-emptive regions.

VIII. Experimental Section

In this paper the theory for limited pre-emptive global fixed task priority scheduling is presented. In order to assess the performance of this scheduling discipline first we examine the relative performance of the three methods of estimating the blocking induced by lower or equal priority workload.

A. Blocking Estimation

We generate 100 sets, with n Q elements. These sets are intended to represent the last non-pre-emptive regions from all tasks in a given taskset. Each last non-pre-emptive region length is a randomly generated value in the range $[0, 300]$. The quantity A_1^k to A_{n-k}^k is upper-bounded for each of these Q sets using the three methods described previously. The estimations are performed for each generated set of Q values starting with priority 1 (i.e. computing A_1^k) until priority level $n-k$ (A_{n-k}^k). The average last non-pre-emptive region length (Q_i) is computed over the 100 task-sets for each priority level i using each of the three methods. The results are presented in Figure 4.

From the results in Figure 4 it is clear that the third estimation mechanism outperforms the first two as expected. The first one is the crudest approximation, its estimations tend to be much more pessimistic than the other two. Whereas the second one, albeit simple enough, provides results that are similar to the third and most complex of the three. As the priority level decreases (i.e. task index increases) the estimations tend to decrease since any subset of k values will necessarily be smaller than or equal to any in a larger set.

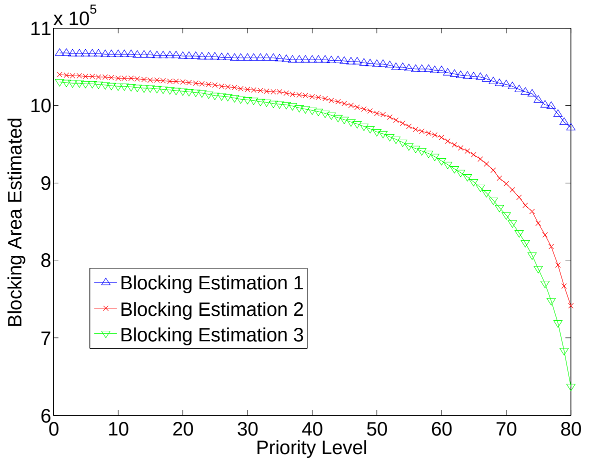


Fig. 4: Blocking Estimations (k=8,n=88).

The two latter methods tend to decrease their estimation faster as the priority level increases since the number of values to chose from decreases whereas the first method, by basing its estimation on the maximum value present in the set will not reduce its estimation as steeply.

B. *Pre-emptions in Simulated Schedules*

In order to assess the performance of the ADS scheduling policy with respect to the observed pre-emptions in a given schedule a simulator was created. Task-sets are randomly generated and the schedule produced by fully pre-emptive global fixed task priority and ADS is generated. In the simulated schedules the number of direct pre-emptions is extracted. Each task-set is randomly generated where $Utot$ is the target total utilization. The individual task utilizations $0 < u_i \leqslant 1$ are obtained by the random fixed sum method [24]. The execution requirement of each task C_i is a uniformly distributed random variable in the interval $[100, 500]$. The relative deadlines of the tasks are computed then as $D_i = \frac{C_i}{u_i}$. The period of each task T_i is equal to the relative deadline ($T_i = D_i$).

The priority assigned to the tasks is the same for both fully pre-emptive and ADS simulations. The heuristic employed to assign priorities is DkC [25]. The schedules are simulated for platforms comprising m cores. In Figures 5a and 5b $m = 2$. Whereas Figures 5c 5d relate to simulations on four processors. The simulations are run for 5000000 time units. The total utilizations of the taskset are varied from 0.1 to m with steps of 0.1 units. At each utilization level 100 random task-sets are generated and their schedules simulated.

Since ADS is compared against the global fully pre-emptive fixed task priority only task-sets which are schedulable by the latter are considered. As a consequence, while computing the last non-pre-emptive regions for each task with Algorithm 4 the minimum Q vector considered is one where all elements are zero. Since the performance of the blocking estimation 1 is the most modest, this was put to use in order to get a sense of the worst-case performance of ADS and to show that even in those circumstances it compares quite favorably against the fully pre-emptive scheduler with respect to run-time pre-emptions.

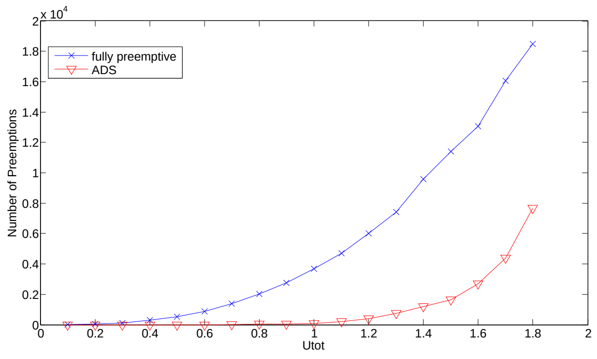
From the presented results it is apparent that a large number of the pre-emptions are removed from the actual schedule. From figures 5a to 5d it is obvious that the number of pre-emptions in ADS tends to decrease with increases in n. This is due to the spread of the available utilization among constituents of the task-set. Consequently tasks will tend to have moderately similar deadlines and execution requirements. This is beneficial for obtaining larger admissible non-pre-emptive regions when compared to the execution time. The number of pre-emptions in both scheduling policies increase with the total utilization of the task-set, still the pre-emption increase in fully pre-emptive tends to be steeper than in the ADS schedule. Since the processors tend to be occupied for larger time intervals it is more likely that newly released jobs will induce a pre-emption.

As the number of processors increase the relative benefits of the ADS policy suffer a mild degradation (comparison between m=4 and m=2). This is due to the poor performance of the blocking estimation mechanism put to use in this simulation effort (ADS estimation 1) as it will severely over-estimate the actual worst-case blocking time tasks will be subject to and as a consequence will lead to smaller non-pre-emptive region lengths. This induces more pre-emptions points in the tasks and hence more possibilities for pre-emptions to occur. To be noted that the blocking estimation 1 and estimation 2 in this case would yield similar results since by taking a top down approach and by assuming that the lower priority non-preemptive regions would be equal to Q_i, the A_i^k estimate is the same for both methods as there would exist only a single distinct non-preemptive region length value which would be mandatorily the maximum. Another shortcoming general to all the blocking estimation mechanisms presented in this work is that these do not take into account the maximum execution requirement of the lower or equal priority tasks. As m increases so does the pessimism involved in the estimation step since the stair-case pattern of blocking is subject to cruder overestimations as the number of steps increases.

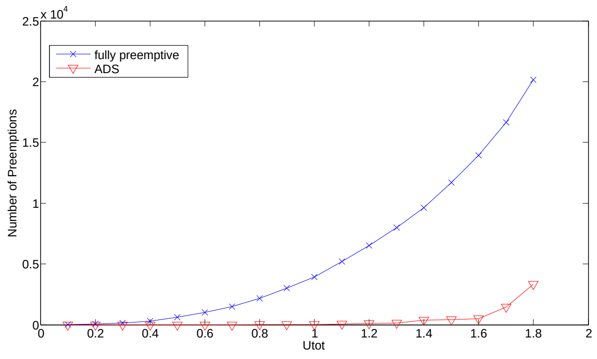
IX. Conclusions and Future Work

In this work we present a novel limited pre-emptive global scheduling policy. The schedulability test is presented with three approaches to estimate the blocking from lower or equal priority non-pre-emptive regions. The new scheduling policy is shown to ensure that a job can be blocked by lower priority workload only before its first dispatch. This compares favorably with the scheduling policy termed RDS which is subject to multiple instances of blocking throughout the execution of a given job. The presented scheduling policy (ADS) dominates global fully pre-emptive fixed task priority and fully non-pre-emptive scheduling with respect to schedulability. As final contributions the blocking estimation mechanisms are compared against each other. Finally, the ADS policy is shown to drastically reduce the number of pre-emptions occurring in the schedule when compared to global fully pre-emptive scheduling.

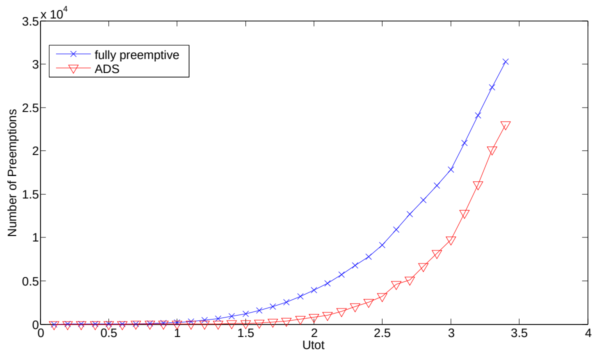
As future work we intend enhance the system model and consider non-negligible pre-emption and migration delay penalties. In this manner we will exploit the limited pre-emptive model in order to decrease the pre-emption and migration delay involved with the scheduling of task-sets upon multiprocessors. The contention at the memory access bus and controller will similarly be modeled and the interference resulting from the contention at this shared resource will be integrated into the non-preemptive schedulability analysis. Furthermore we wish to devise clustering techniques so as to alleviate the ADS



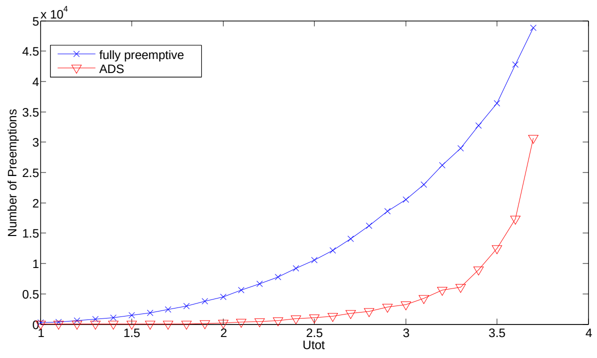
(a) m=2,n=20



(b) m=2,n=32



(c) m=4,n=32



(d) m=4,n=64

Fig. 5: Observed Pre-emptions in Simulated Schedules.

pessimism when large quantities of processors comprise the execution platform.

References

[1] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st conference on Computing Frontiers*, 2004.

[2] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A performance counter architecture for computing accurate CPI components," *SIGOPS Operating*, 2006.

[3] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Survey*, vol. 43, no. 4, pp. 35:1–35:44, Oct 2011.

[4] W. River, "Vxworks platforms," http://www.windriver.com/products/product-notes/PN_VxWorks_3_8_Jan_2010.pdf.

[5] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," in *OSPERT*, 2010.

[6] T. Baker, "Multiprocessor edf and deadline monotonic schedulability analysis," in *RTSS*, 2003.

[7] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *RTSS*, 2007.

[8] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *RTSS*, 2007.

[9] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *RTSS*, 2009.

[10] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011.

[11] A. Burns, "Preemptive priority-based scheduling: an appropriate engineering approach," in *Advances in real-time systems*, S. H. Son, Ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[12] R. Bril, J. Lukkien, and W. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited," in *ECRTS*, 2007.

[13] R. Bril, J. Lukkien, , and W. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, vol. 42, pp. 63–119, 2009. [Online]. Available: http://www.springerlink.com/content/f05r404j63424h27/fulltext.pdf

[14] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *ECRTS*, 2005.

[15] G. Yao, G. Buttazzo, and M. Bertogna, "Bounding the maximum length of non-preemptive regions under fixed priority scheduling," *RTCSA 2009*.

[16] ——, "Feasibility analysis under fixed priority scheduling with limited preemptions," *Real-Time Systems*, vol. 47, no. 3, 2011.

[17] ——, "Comparative evaluation of limited preemptive methods," in *ETFA 2010*.

[18] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo, "Optimal selection of preemption points to minimize preemption overhead," in *RTSS*, 2011.

[19] N. Guan, W. Yi, Q. Deng, Z. Gu, and G. Yu, "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling," *Journal of Systems Architecture*, vol. 57, no. 5, pp. 536 – 546, 2011.

[20] J. Lee and K. Shin, "Controlling preemption for better schedulability in multi-core systems," in *RTSS 2012*.

[21] R. I. Davis, A. Burns, J. Marinho, V. Nélis, S. M. Petters, and M. Bertogna, "Global fixed priority scheduling with deferred pre-emption," in *RTCSA*, 2013.

[22] R. I. Davis and M. Bertogna, "Optimal fixed priority scheduling with deferred pre-emption," in *RTSS*, 2012.

[23] J. Marinho, V. Nélis, S. M. Petters, M. Bertogna, and Rob, "Limited preemption global fixed task priority," CISTER/INESC-TEC, Rua Alfredo Allen 535, 4200-135 PORTO, Portugal, Tech. Rep. CISTER-

TR-130505, may 2013. [Online]. Available: www.cister.isep.ipp.pt/docs/767/

[24] P. Emberson, R. Stafford, and R. Davis, “Techniques for the synthesis of multiprocessor tasksets,” in *WATER*, 2010.

[25] R. Davis and A. Burns, “Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” in *RTSS*, 2009.