Optimal Fixed Priority Scheduling with Deferred Pre-emption

Robert I. Davis
Real-Time Systems Research Group,
Department of Computer Science,
University of York, York, UK.
rob.davis@york.ac.uk

Marko Bertogna
Algorithmic Research Group,
Department of Mathematics,
University of Modena, Italy
marko.bertogna@unimore.it

***Abstract*— The schedulability of systems using fixed priority pre-emptive scheduling can be improved by the use of non-pre-emptive regions at the end of each task's execution; an approach referred to as deferred pre-emption. Choosing the appropriate length for the final non-pre-emptive region of each task is a trade-off between improving the worst-case response time of the task itself and increasing the amount of blocking imposed on higher priority tasks. In this paper we present an optimal algorithm for determining both the priority ordering of tasks and the lengths of their final non-pre-emptive regions. This algorithm is optimal for fixed priority scheduling with deferred pre-emption, in the sense that it is guaranteed to find a schedulable combination of priority ordering and final non-pre-emptive region lengths if such a schedulable combination exists.**

Keywords-real-time scheduling; schedulability analysis; fixed priority; deferred pre-emption; optimal priority assignment; non-pre-emptive scheduling.

I. INTRODUCTION

A common misconception with regard to fixed priority scheduling of sporadic tasks on a single processor is that fully pre-emptive scheduling is the best approach to take in terms of taskset schedulability. Fixed priority non-pre-emptive scheduling (FPNS) and fixed priority pre-emptive scheduling (FPPS) are however incomparable; there are tasksets that are schedulable under FPNS that are not schedulable under FPPS and vice-versa.

In the literature, the term *fixed priority scheduling with deferred pre-emption* (FPDS) has been used to refer to a variety of different techniques by which pre-emptions may be deferred for some period of time after a higher priority task becomes ready. These are discussed in Section II. In this paper, we assume a form of fixed priority scheduling with deferred pre-emption where each task has a final non-pre-emptive region. If this region is of the minimum possible length[1] for all tasks, then we have fixed priority pre-emptive scheduling, whereas if the final non-pre-emptive region constitutes all of the task's execution time then we have fixed priority non-pre-emptive scheduling. Thus FPDS can be viewed as a superset of both FPPS and FPNS. FPDS can therefore potentially schedule any taskset that is schedulable under FPPS or FPNS; in other words FPDS dominates both FPPS and FPNS.

With FPDS, there are two key parameters that affect taskset schedulability: the priority assigned to each task, and the length of each task's final non-pre-emptive region. The length of the final region affects both the schedulability of the task itself, and the schedulability of tasks with higher priorities. This is a trade-off, increasing the length of the final non-pre-emptive region can improve schedulability for the task itself by reducing the number of times it can be pre-empted by higher priority tasks, but potentially increases blocking on higher priority tasks reducing their schedulability.

In this paper, we introduce an optimal algorithm for FPDS. This *Final Non-pre-emptive Region and Priority Assignment* (*FNR-PA*) algorithm is optimal in the sense that it is guaranteed to find a combination of priority assignment and final non-pre-emptive region lengths that result in a schedulable system under FPDS whenever such a schedulable combination of these parameters exists. Stated otherwise, the FNR-PA algorithm is able to find a schedulable solution for *any* taskset that is feasible under FPDS. The algorithm relies on finding the minimum final non-pre-emptive region length for each task that ensures its schedulability. This value may be found via binary search; however, we also derive a more efficient analytical method. The algorithm takes a greedy approach to priority assignment and is therefore tractable in terms of the number of task schedulability tests that it requires.

The remainder of the paper is organised as follows. Section II describes the background research which we build upon. Section III describes the system model, terminology and notation used. Section IV recapitulates schedulability analysis for fixed priority scheduling with deferred pre-emption. Section V provides formal definitions of the problems addressed, and derives algorithms to solve them. Section VI describes an analytical method for determining the minimum final non-pre-emptive region length commensurate with task schedulability. Section VII provides an experimental evaluation, comparing the effectiveness of FPDS with optimal priority and final non-pre-emptive region length assignment, with that of FPPS and FPNS. Finally, Section VIII concludes with a summary and directions for future work.

[1] The minimum possible length of a non-pre-emptive region is 1 rather than 0, as we assume a discrete time model and tasks cannot be pre-empted during a processor clock cycle.

II. BACKGROUND RESEARCH

In 1973, Liu and Layland [32] considered FPPS of synchronous tasksets comprising independent periodic tasks, with bounded execution times, deadlines equal to their periods (referred to as *implicit-deadlines*), and a common release time. Liu and Layland showed that *rate monotonic* priority ordering (RMPO) is the optimal fixed priority ordering for such tasksets.

Research into real-time scheduling during the 1980's and early 1990's focused on lifting many of the restrictions of the Liu and Layland task model. Task arrivals were permitted to be sporadic, with known minimal inter-arrival times, (still referred to as periods), and task deadlines were permitted to be less than or equal to their periods (referred to as *constrained deadlines*) or less than, equal to, or greater than their periods (referred to as *arbitrary deadlines*).

In 1982, Leung and Whitehead [31] showed that *deadline monotonic*[2] priority ordering (DMPO) is the optimal fixed priority ordering for constrained-deadline tasksets. Exact schedulability tests for FPPS of constrained-deadline tasksets were introduced by Joseph and Pandya in 1986 [28], Lehoczky et al. in 1989 [29], and Audsley et al. in 1993 [4].

In 1990, Lehoczky [30] showed that DMPO is not optimal for tasksets with arbitrary deadlines; however, an optimal priority ordering for such tasksets can be determined in at most $n(n+1)/2$ task schedulability tests using Audsley's optimal priority assignment (OPA) algorithm[3] [3], [5]. Exact schedulability tests for tasksets with arbitrary deadlines were developed by Lehoczky [30] in 1990, and Tindell et al. [34] in 1994.

In 1996, George et al. [27] derived an exact schedulability test for FPNS based on the approach of Tindell et al. [34] for the pre-emptive case. George et al. showed that unlike in the pre-emptive case, DMPO is not optimal for constrained-deadline tasksets scheduled by FPNS. Further, they showed that Audsley's optimal priority assignment algorithm [5] is applicable, and can also be used to determine an optimal priority ordering for tasksets with arbitrary-deadlines in the non-pre-emptive case.

In 2008, Davis and Burns [23] provided linear response time upper bounds and hence sufficient schedulability tests for FPPS, FPNS and FPDS. Davis et al. [24] also showed how to use response time lower bounds to improve the efficiency of the exact schedulability tests.

Two different models of fixed priority scheduling with deferred pre-emption have been developed in the literature.

In the *fixed* model, introduced by Burns in 1994 [19], the location of each non-pre-emptive region is statically determined prior to execution. Pre-emption is only permitted at pre-defined locations in the code of each task, referred to as *pre-emption points*. This method is also referred to as *co-operative scheduling*, as the tasks co-operate, providing re-scheduling / pre-emption points to improve schedulability.

In the *floating* model [6], [36], an upper bound is given on the length of the longest non-pre-emptive region of each task. However, the location of each non-pre-emptive region is not known a priori and may vary at run-time, for example under the control of the operating system.

Exact schedulability analysis for the fixed model was derived by Bril et al. in 2009 [15]. Recently, Bertogna et al. integrated pre-emption costs and cache related pre-emption delays (CRPD) into analysis of the fixed model, considering both fixed [11] and variable [12] pre-emption costs.

In 2011, Bertogna et al. [13], derived a method of computing the optimal length of the final non-pre-emptive region of each task in order to maximize schedulability under FPDS for a given priority assignment.

Alternative approaches to limiting pre-emption include *Pre-emption Thresholds* (FPTS) [35], [33] and *Non-pre-emption Groups* [21] in which each task has a base priority at which it competes for the processor, and a *threshold* priority at which it executes, thus limiting pre-emption to those tasks with a base priority higher than the threshold. In [35], [33] Saksena and Wang attempted to derive an integrated approach to priority and pre-emption threshold assignment, but did not succeed in finding an optimal algorithm with less than exponential complexity. Research by Bril et al. [16] in 2012 combines the ideas of deferred pre-emption and pre-emption thresholds, generalising both into a single scheme with pre-emption thresholds between a set of sub-jobs which execute non-pre-emptively.

For further information on limited pre-emption scheduling the reader is referred to the survey in [17].

III. SYSTEM MODEL, TERMINOLOGY AND NOTATION

In this paper, we consider the fixed priority scheduling of a set of sporadic tasks (or *taskset*) on a single processor. Each taskset comprises a static set of n tasks ($\tau_1..\tau_n$), where n is a positive integer. We assume that the index i of task τ_i represents the task priority, hence τ_1 has the highest priority, and τ_n the lowest. We assume a discrete time model, where all task parameters are assumed to be positive integers.

We use the notation $hp(i)$ (and $lp(i)$) to mean the set of tasks with priorities higher than (lower than) i, and the notation $hep(i)$ (and $lep(i)$) to mean the set of tasks with priorities higher than or equal to (lower than or equal to) i.

Each task τ_i is characterized by its bounded worst-case execution time C_i, minimum inter-arrival time or period T_i, and relative deadline D_i. Each task τ_i therefore gives rise to a potentially unbounded sequence of invocations (or *jobs*), each of which has an execution time upper bounded by C_i, an arrival time at least T_i after the arrival of its previous job, and an absolute deadline that is D_i after its arrival.

In an *implicit-deadline* taskset, all tasks have $D_i = T_i$. In a *constrained-deadline* taskset, all tasks have $D_i \le T_i$, while in an *arbitrary-deadline* taskset, task deadlines are independent of their periods.

[2] *Deadline monotonic* priority ordering assigns priorities in order of task deadlines, such that the task with the shortest deadline is given the highest priority.

[3] This algorithm is optimal in the sense that it finds a schedulable priority ordering whenever such an ordering exists.

The *utilisation* U_i of a task τ_i is given by its execution time divided by its period ($U_i = C_i / T_i$). The total utilisation U of a taskset is the sum of the utilisations of all of its tasks.

Under FPDS, each task is assumed to have a final non-pre-emptive region of length F_i in the range $[1, C_i]$. Determining an appropriate value for the length of this region is assumed to be part of the scheduling problem rather than a characteristic of the task.

We assume that tasks may make mutually exclusive access to shared resources according to the Stack Resource Policy (SRP) [8]. We use B_l^i to denote the longest time that a task $\tau_l \in lp(i)$ may execute while holding a resource that is shared with a task of priority i or higher. B_l^i therefore represents the longest time for which task τ_l can execute at priority i or higher due to the operation of the SRP. We assume that any resource access occurring within the final non-pre-emptive region of a task is properly nested, i.e. wholly included within that region. We note that this may place constraints on the specific values that the length of the final non-pre-emptive region of each task may take.

The following assumptions are made about the behaviour of the tasks. The arrival times of the tasks are independent and unknown a priori; hence the tasks may share a common release time. Each task is released, i.e. becomes ready to execute, as soon as it arrives. The tasks do not voluntarily suspend themselves.

The *worst-case response time* R_i of a task is given by the longest possible time from release of the task until it completes execution. Thus task τ_i is schedulable if and only if $R_i \le D_i$, and a taskset is schedulable if and only if $\forall i \quad R_i \le D_i$. A *critical instant* task τ_i is a scenario or pattern of task releases that leads to the worst-case response time for task τ_i.

Under FPDS, at any given time the highest priority ready task is selected for execution by the processor. Note both final non-pre-emptive regions and resource accesses according to SRP are assumed to be implemented by manipulating task priorities. Thus a task executing a final non-pre-emptive region has the highest priority and so will not be pre-empted.

A taskset is said to be *schedulable* with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the taskset can be scheduled by the algorithm without any missed deadlines.

A priority assignment policy P is said to be *optimal* with respect to some class of tasksets (e.g. arbitrary-deadline), and some type of fixed priority scheduling algorithm (e.g. FPPS, FPNS, or FPDS) if there are no tasksets in the class that are schedulable under the scheduling algorithm using any other priority ordering policy, that are not also schedulable using the priority assignment determined by policy P.

A fixed priority scheduling algorithm A is said to *dominate* another fixed priority scheduling algorithm B if there are tasksets that can be scheduled under algorithm A, but cannot be scheduled under algorithm B, and all of the tasksets that are schedulable under algorithm B are also schedulable under algorithm A. If there are tasksets that are schedulable under algorithm A, but not under algorithm B, and vice-versa, then the two algorithms are said to be *incomparable*. If both algorithms can schedule precisely the same tasksets then they are said to be *equivalent*.

A scheduling algorithm is said to be *sustainable* [7], [20] with respect to a system model, if and only if schedulability of any taskset compliant with the model implies schedulability of the same taskset modified by: (i) decreasing execution times, (ii) increasing periods or inter-arrival times, and (iii) increasing deadlines. Similarly, a schedulability test is referred to as *sustainable* if these changes cannot result in a taskset that was previously deemed schedulable by the test becoming unschedulable. We note that the modified taskset may not necessarily be deemed schedulable by the test. A schedulability test is referred to as *self-sustainable* [9] if such a modified taskset is always deemed schedulable by the test.

IV. Recapitulation of Schedulability Analysis for FPDS

We now recapitulate schedulability analysis for fixed priority scheduling with deferred pre-emption for sporadic tasksets with arbitrary deadlines, based on the work of Bril et al. [15]. In order to deal with the discrete time domain assumed in this paper, the results presented in [15] are rephrased according to the notation adopted in [17]. We also make simple extensions to the analysis to account for blocking due to resource accesses.

First, we introduce the concepts of priority level-i active period, and Δ-critical instant, which are fundamental to the analysis of FPDS.

The term *priority level-i active period* refers to a continuous period of time $[t_1, t_2)$ during which tasks, of priority i or higher, that were released at the start of the active period at t_1, or during the active period but strictly before its end at t_2, are either executing or ready to execute.

A Δ-*critical instant* for a task τ_i occurs when task τ_i is released simultaneously with all higher priority tasks, and subsequent releases of task τ_i and higher priority tasks occur after the minimum permitted time intervals. Further, the minimum possible amount of time Δ prior to this simultaneous release, a lower priority task τ_k enters its final non-pre-emptive region or begins accessing a resource shared with a task of priority i or higher. Note that due to the integer time model considered in this paper, the discrete time granularity Δ is one time unit.

Bril et al. [15] showed that for FPDS, the longest response time of a task τ_i occurs for some job of that task within the priority level-i active period starting at a Δ-*critical instant*. Lemma 3 in [15] states that the worst-case length of a priority level-i active period A_i is given by the minimum solution to the following fixed point iteration:

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j \qquad (1)$$

Iteration starts with an initial value A_i^0 guaranteed to be no larger than the minimum solution, for example $A_i^0 = C_i$, and ends when $A_i^{m+1} = A_i^m$. In (1) the term B_i is the longest time that task τ_i can be blocked from executing by lower priority tasks, and is given by:

$$B_i = \max(B_i^{RES}, B_i^{FNR}) \quad (2)$$

$$B_i^{RES} = \max_{\forall l \in lp(i)} (B_l^i - 1), \quad B_i^{FNR} = \max_{\forall l \in lp(i)} (F_l - 1)$$

where B_i^{RES} and B_i^{FNR} are respectively the blocking factors at priority *i* due to resource locking and final non-pre-emptive regions. In (2) the maximum of an empty set is assumed to be zero.

The number of jobs G_i of task τ_i in the priority level-*i* active period is given by:

$$G_i = \left\lceil \frac{A_i}{T_i} \right\rceil \quad (3)$$

The *start time* $W_{i,g}^{NP}$ of the final non-pre-emptive region of job *g* (where *g* = 0 is the first job) of task τ_i measured with respect to the start of the Δ-critical instant is given by the minimum solution to the following fixed point iteration:

$$w_{i,g}^{m+1} = B_i + (g+1)C_i - F_i + \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{w_{i,q}^m}{T_j} \right\rfloor + 1 \right) C_j \quad (4)$$

Iteration starts with an initial value $w_{i,g}^0$ guaranteed to be no greater than $W_{i,g}^{NP}$, typically $w_{i,g}^0 = B_i + (g+1)C_i - F_i$, and ends when either $w_{i,g}^{m+1} = w_{i,g}^m$ in which case $W_{i,g}^{NP} = w_{i,g}^{m+1}$, or when: $w_{i,g}^{m+1} + F_i - gT_i > D_i$ in which case job *g*, and hence task τ_i is unschedulable.

To find the worst-case response time, the start times of the final non-pre-emptive regions $W_{i,g}$ need to be calculated for jobs $g = 0,1,2,3,...G_i - 1$. The worst-case response time of task τ_i is then given by:

$$R_i = \max_{\forall g=0,1,2...G_i-1} (W_{i,g}^{NP} + F_i - gT_i) \quad (5)$$

Task τ_i is schedulable provided that $R_i \le D_i$.

Corollary 1: *Sustainability of task schedulability with respect to an increase in the length of its final non-pre-emptive region.* From (4) and (5), the worst-case response time of task τ_i is monotonically non-increasing with respect to increases in the length of its final non-pre-emptive region F_i. Stated otherwise, increasing the length of the final non-pre-emptive region of a task cannot result in that task becoming unschedulable if it was previously schedulable.

A. Example of FPDS

We now provide an example of fixed priority scheduling with deferred pre-emption. The example is based on the taskset in Table I, and is derived from an example for fixed priority non-pre-emptive scheduling given in [25]. In this example, we assume that the tasks are independent.

First let us consider which priority assignments might lead to a schedulable system. The deadline of 175 for task τ_A means that it is trivially unschedulable at anything other than the highest priority level, irrespective of whether we use FPPS, FPNS, or FPDS, and whatever the lengths of any final non-pre-emptive regions. We must therefore place task τ_A at the highest priority.

TABLE I: TASK PARAMETERS

Task	Execution time	Period	Deadline
τ_A	100	250	175
τ_B	100	400	300
τ_C	100	350	325

Now let us consider assigning task τ_C the lowest priority. Corollary 1 tells us that the best schedulability for task τ_C is achieved if it is completely non-pre-emptive, i.e. $F_C = C_C = 100$. From (1), the length of the priority level-3 active period is 700, and so we must examine the response times of the first two jobs of task τ_C, starting from a Δ-critical instant. This sequence of task execution is as shown in Figure 1, assuming that task τ_C executes non-pre-emptively. The second job of task τ_C misses its deadline at time 675; hence task τ_C cannot be given the lowest priority.

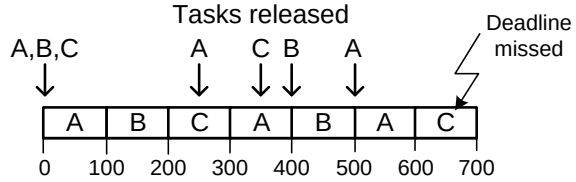

Figure 1: Priority level-3 active period. Task C at the lowest priority level.

We observe that (τ_A , τ_B , τ_C) represents deadline monotonic priority order (DMPO). So DMPO cannot make this taskset schedulable, irrespective of what lengths we might choose for the final non-pre-emptive regions.

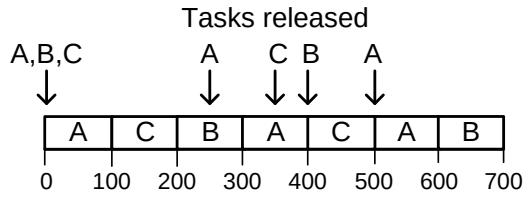

Figure 2: Priority level-3 active period. Task B at the lowest priority level.

Now let us consider placing task τ_B at the lowest priority level, i.e. priority order (τ_A, τ_C, τ_B). If we set the length of the final non-pre-emptive region of task τ_B to be $F_B = 51$, then the priority level-3 active period is as shown in Figure 2, and the two jobs of task τ_B both meet their deadlines. Further, the worst-case response times of task τ_A and task τ_C are 150, and 250 respectively. Thus with a priority ordering (τ_A , τ_C , τ_B) and final non-pre-emptive region lengths of $F_A = 1$, $F_C = 1$, and $F_B = 51$ the taskset is schedulable using FPDS.

Assuming FPPS with optimal (deadline monotonic) priority ordering, then task τ_C misses its deadline at 325 due to pre-emption by task τ_A at 250; hence, the taskset is

unschedulable under FPPS with any priority ordering [31]. The taskset is also trivially unschedulable under FPNS as task τ_A cannot tolerate being blocked for more than 75 and the execution times of tasks τ_B and τ_C are 100.

This example illustrates the dominance, rather than equivalence, of FPDS over FPPS and FPNS. It also serves to show that deadline monotonic priority ordering is not optimal for FPDS.

V. Optimal Fixed Priority Scheduling with Deferred Pre-emption

In the previous section, we showed that deadline monotonic priority ordering is not optimal for FPDS, even for tasks with constrained deadlines. In this section, we pose two key problems relating to the assignment of priorities and final non-pre-emptive region lengths under FPDS. We then derive tractable optimal algorithms that solve these problems.

Note we recognise that there may be constraints on the permissible set of values for the length of the final non-pre-emptive region of each task. In this section when we refer to values for the length of the final non-pre-emptive region of a task, we mean *valid* values. We return to this point towards the end of Section VI.

Problem 1: *Final Non-pre-emptive Region length Problem* (*FNR Problem*). For a given taskset complying with the task model described in Section III, and a given priority ordering *X*, find a value for the length of the final non-pre-emptive region of each task such that the taskset is schedulable under FPDS.

Definition 1: An algorithm *A* is said to be *optimal* for the *FNR Problem* if there are no taskset / priority assignment combinations that are schedulable under FPDS with some set of values for the lengths of the final non-pre-emptive regions of each task, that are not also schedulable using the set of values for the lengths of final non-pre-emptive regions determined by algorithm *A*.

Problem 2: *Final Non-pre-emptive Region Length and Priority Assignment Problem* (*FNR-PA Problem*). For a given taskset complying with the task model described in Section III, find both (i) a priority assignment, and (ii) a value for the length of the final non-pre-emptive region of each task that makes the taskset schedulable under FPDS.

Definition 2: An algorithm *B* is said to be *optimal* for the *FNR-PA Problem* if there are no tasksets compliant with the task model that are schedulable under FPDS with some priority assignment *X* and some set of values for the lengths of the final non-pre-emptive regions of each task, that are not also schedulable using the priority assignment and set of values for the lengths of the final non-pre-emptive regions determined by algorithm *B*.

We now derive tractable algorithms that solve the FNR and FNR-PA problems. In particular, the solution to the FNR-PA problem provides optimal fixed priority scheduling with deferred pre-emption.

First we introduce some additional notation and a number of corollaries from the schedulability analysis for FPDS given in Section IV.

For a given taskset and priority ordering *X*, we use $hep(k, X)$ to mean the set of tasks with priority higher than or equal to *k*. Similarly, $hp(k, X)$ is the set of tasks with priorities strictly higher than *k*, and $lp(k, X)$ is the set of tasks with priorities strictly lower than *k*, in priority order *X*.

We use $F(k, X)$ to denote the length of the final non-pre-emptive region of the task at priority *k* in priority order *X*, and similarly, $B(k, X)$ to mean the blocking factor at priority level *k*. Where it is unnecessary to explicitly refer to the priority ordering, we use a short form of this notation e.g. $hep(k)$, $hp(k)$, $lp(k)$, $F(k)$, and $B(k)$.

We observe the following corollaries which follow directly from (4) and (5).

Corollary 2: *Sustainability [7] of task schedulability under FPDS with respect to a decrease in the blocking factor*. A task that is schedulable at priority level *k* with a blocking factor $B(k)$ due to a set of lower priority tasks $lp(k)$ remains schedulable when the blocking factor is reduced (e.g. by reducing the length of the final non-pre-emptive region of one or more lower priority tasks) and the sets $lp(k)$ and $hp(k)$ of lower and higher priority tasks remain unchanged.

Corollary 3: The schedulability under FPDS of a task at priority *k* with a final non-pre-emptive region of length $F(k)$ depends on the set of higher priority tasks $hp(k)$, but is independent of the relative priority ordering of those tasks.

Corollary 4: The schedulability under FPDS of a task at priority *k* with a final non-pre-emptive region of length $F(k)$ depends on the set $lp(k)$ of lower priority tasks in respect of the blocking factor; however, there is no dependency on the relative priority ordering of the lower priority tasks.

Corollary 5: For a given set of higher priority tasks $hp(k)$, the minimum value for the length $F(k)$ of the final non-pre-emptive region of task τ_k consistent with that task remaining schedulable under FPDS is a monotonically non-decreasing function of the blocking factor $B(k)$. Stated otherwise, a larger blocking factor at priority *k* cannot result in a smaller minimum length for the final non-pre-emptive region of the task at that priority level.

Note, an analytical technique for determining the smallest final non-pre-emptive region length for each task is described in Section VI.

```
for each priority level k, lowest first {
    determine the smallest value for the final
    non-pre-emptive region length F(k) such that
    the task at priority k is schedulable.
    Set the length of the final non-pre-emptive
    region of the task to this value.
}
```

Algorithm 1: FNR Algorithm

Theorem 1: The FNR algorithm (Algorithm 1) is *optimal* for the FNR problem (see Problem 1 and Definition 1).

Proof: We assume (for contradiction) that there exists a taskset τ and priority ordering X that is schedulable with some set of final non-pre-emptive region lengths $F'(k, X)$ for k = 1 to n, for which the FNR algorithm fails to determine a set of non-pre-emptive region lengths $F(k, X)$ for k = 1 to n, that also results in a schedulable system.

Let $B'(k, X)$ be the blocking factor at priority k with the schedulable set of final non-pre-emptive region lengths, and $B(k, X)$ be the blocking factor at priority k with the set of final non-pre-emptive region lengths computed by the FNR Algorithm. At each priority level, we will show that $F(k, X) \le F'(k, X)$ and hence from (2) that $B(k, X) \le B'(k, X)$ thus proving via Corollary 2 *sustainability of task schedulability with respect to blocking factors* that the taskset is schedulable with priority ordering X and the final non-pre-emptive region lengths determined by the FNR Algorithm, thus contradicting the original assumption.

The proof is by induction over each priority level k from n to 1.

Initial step: At the lowest priority level, n we have $B(n, X) = B'(n, X) = 0$. At priority n, the FNR Algorithm (Algorithm 1) computes the minimum schedulable final non-pre-emptive region length $F(n, X)$ for task τ_n, hence $F(n, X) \le F'(n, X)$.

Inductive step: We assume that at priority k, $B(k, X) \le B'(k, X)$ and $F(k, X) \le F'(k, X)$. As the resource accesses by the task at priority k are unchanged, then from (2) we have $B(k-1, X) \le B'(k-1, X)$ and thus via Corollary 5, $F(k-1, X) \le F'(k-1, X)$.

Iterating over all of the priority levels shows that for all k from n to 1, $B(k, X) \le B'(k, X)$ and so by Corollary 2, the taskset is schedulable with the set of final non-pre-emptive region lengths $F(k, X)$ obtained by Algorithm 1 □

```
for each priority level k, lowest first {
   for each unassigned task τ {
        determine the smallest value for the
        final non-pre-emptive region length F(k)
        such that task τ is schedulable at
        priority k, assuming all other
        unassigned tasks have higher priorities.
        Record as task Z the unassigned task
        with the minimum value for the length of
        its final non-pre-emptive region F(k).
   }
   if no tasks are schedulable at priority k {
        return unschedulable
   }
   else {
        assign priority k to task Z and use the
        value of F(k) as the length of its final
        non-pre-emptive region.
   }
}
return schedulable
```

Algorithm 2: FNR-PA Algorithm

Corollary 6: For a given taskset and fixed priority ordering X, that is schedulable under FPDS with some set of final non-pre-emptive region lengths, Algorithm 1 minimises the final non-pre-emptive region length of every task, and hence minimises the blocking factor at every priority level.

Theorem 2: The Final Non-pre-emptive Region Priority Assignment (FNR-PA) algorithm (Algorithm 2) is *optimal* for the FNR-PA problem (see Problem 2 and Definition 2).

Proof: We assume (for contradiction) that there exists a taskset τ with a priority ordering X and a set of final non-pre-emptive region lengths $F(k, X)$ for k = 1 to n that result in a schedulable system. Without loss of generality, we assume that these final non-pre-emptive region lengths are the smallest possible values for this priority assignment – see Theorem 1. Further, we assume that the FNR-PA algorithm is unable to find a schedulable priority ordering and set of final non-pre-emptive region lengths for taskset τ.

For the purposes of the proof, we will refer to the schedulable priority ordering X as X_n. We will iteratively transform X_n into X_{n-1} .. X_1, where X_1 is the same priority order as the complete priority ordering P generated by the FNR-PA algorithm. The transformation will be such that the taskset remains schedulable thus proving the theorem via contradiction. Further, we will show that the FNR-PA algorithm is able to generate a complete priority ordering P.

We use k as an iteration count and also the priority level that we will transform. Thus k counts down from an initial value of n to 1, and represents the priority level at which the FNR-PA algorithm assigns a task. We note that as a result of the transformations, the tasks at priority levels lower than k become the same in both X_k and P, hence once iteration is complete, X_1 and P represent the same priority ordering.

We show that at each priority level k, the FNR-PA algorithm can always find at least one schedulable task among those tasks of priority k or higher in X_k (i.e. in $hep(k, X_k) = hep(k, P)$). Thus it can find a task to assign to priority k.

On iteration k, the FNR-PA algorithm examines all of the tasks in $hep(k, X_k)$ (i.e. the tasks of priority k or higher in X_k). Of those tasks, it selects the one that is schedulable at priority k, with the minimum length final non-pre-emptive region, and assigns it to that priority level.

The transformation of priority ordering X_k is as follows: First we find the priority level i in X_k of the task that the FNR-PA algorithm selects. We refer to this task as τ_k, as the FNR-PA algorithm will assign it to priority level k. Note that as the tasks of lower priority than k are the same in both X_k and P, priority level i must be either higher than or equal to k.

There are two cases to consider:

1. Task τ_k is at priority k in both P and X_k, in which case no transformation is required on this iteration, and so X_{k-1} is identical to X_k and therefore a schedulable priority ordering.
2. Task τ_k is at a higher priority i in X_k. In this case, we

form priority ordering X_{k-1} by modifying X_k as follows: Task τ_k is moved down in priority from priority level *i* to priority level *k*, and the tasks at priority levels *i*+1 to *k* are all moved up one priority level, as illustrated in Figure 3.

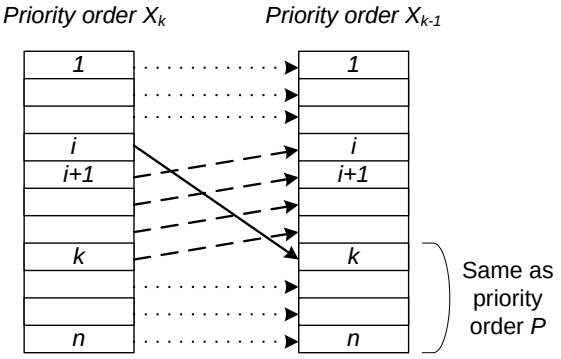

Figure 3: Transformation of priority order

Comparing the tasks in priority order X_{k-1} with their counterparts in X_k. There are effectively four groups of tasks to consider:

1. $\tau_l \in lp(k, X_{k-1})$: Each of these tasks τ_l is assigned the same priority in both X_k and X_{k-1}, and as $hep(k, X_{k-1}) = hep(k, X_k)$, they are subject to interference from the same set of higher priority tasks. Thus these tasks remain schedulable with unchanged final non-pre-emptive section lengths determined as per priority ordering X_k.
2. τ_k: Task τ_k is at priority level *i* in X_k and at the lower priority level *k* in X_{k-1}: We know from the FNR-PA algorithm that τ_k is schedulable at priority *k* and tolerates the minimum value for the final non-pre-emptive region length at priority *k* of any of the tasks in $hep(k, X_k)$. In particular, τ_k tolerates a final non-pre-emptive region length that is at least as short as that of the task at priority *k* in X_k.
3. $\tau_m \in hp(k, X_{k-1}) \cap lep(i, X_{k-1})$: These tasks retain the same partial order but are shifted up one priority level in X_{k-1}. The only difference in the schedulability of a task τ_m in priority order X_{k-1} as compared to priority order X_k is that in the former case, τ_m is subject to interference from task τ_k (which is at the higher priority *i* in X_k) whereas in priority order X_{k-1}, τ_m is subject to blocking from task τ_k (which is at the lower priority level *k* in X_{k-1}) – see Figure 3. Any increasing in the blocking factor at priority *m* due to τ_k being at a lower priority in X_{k-1} cannot exceed its execution time C_k, by contrast the reduction in interference is at least C_k. Thus each task $\tau_m \in hp(k, X_{k-1}) \cap lep(i, X_{k-1})$ remains schedulable under priority order X_{k-1} with its final non-pre-emptive region length set as per priority ordering X_k. Without loss of generality, we assume that the minimum length of each of these final non-pre-emptive regions is re-computed for priority ordering X_{k-1} using Algorithm 1 (i.e. lowest priority first). These values cannot increase with respect to the values for the same tasks in priority ordering X_k (Theorem 1).
4. $\tau_h \in hp(i, X_{k-1})$: These tasks are assigned the same priorities in both X_k and X_{k-1} and as $lep(i, X_{k-1}) = lep(i, X_k)$ they have the same set of lower priority tasks in each case. We now consider the set of lower priority tasks $lep(i, X_{k-1})$. As shown in the above three paragraphs, the only one of these tasks that may have a longer final non-pre-emptive region in priority order X_{k-1} is task τ_k; however, we know that in priority order X_{k-1} task τ_k has a final non-pre-emptive region that is no longer than that of the task at priority *k* in X_k, which is also in the set $lep(i, X_k)$. Thus the blocking factors at all priority levels higher than *i* cannot have increase due to the transformation of the priority order. (The contribution to these blocking factors due to resource locking by the tasks in $lep(i, X_k)$ is the same in both cases, and the contribution due to final non-pre-emptive regions is no larger). Hence all of the tasks $\tau_h \in hp(i, X_{k-1})$ remain schedulable with their final non-pre-emptive region lengths set as per priority ordering X_k. Without loss of generality, we again assume that the minimum lengths of their final non-pre-emptive regions are re-computed for priority ordering X_{k-1} using Algorithm 1 (i.e. lowest priority first). These values cannot increase with respect to the values for the same tasks in priority ordering X_k (Theorem 1).

For each of the four groups of tasks, all of the tasks remain schedulable in priority ordering X_{k-1} with the revised final non-pre-emptive region lengths.

A total of *n* iterations of the above procedure, for values of *k* from *n* down to 1, is sufficient to transform any arbitrary priority ordering *X* into the priority ordering *P*, generated by the FNR-PA algorithm. Further, at each step, the FNR-PA algorithm is able to identify a schedulable task, thus resulting in a schedulable priority assignment and set of final non-pre-emptive region lengths □

We note that the proof technique employed above is similar to that used in the proof of robust priority assignment algorithms in [22], [25].

Theorem 3: For any taskset where there exists a priority ordering and a set of final non-pre-emptive region lengths that is schedulable under FPDS, the FNR-PA algorithm results in a blocking factor B_i^{FNR} from final non-pre-emptive regions at *every* priority level *i* that is no larger than that obtained with any other schedulable priority and final non-pre-emptive region length assignment.

Proof: Let $F^{\max}(k, X)$ denote the length of the longest final-non-pre-emptive region of any task of priority *k* or lower in priority order *X* (i.e. in $lep(k, X)$). Recall that the length of the final non-pre-emptive region of the task at priority *k* in priority order *X* is denoted by $F(k, X)$.

The four numbered paragraphs in the proof of Theorem 2 describe the priority transformation between intermediate priority orderings X_{k-1} and X_k for tasks in four subsets of

the overall priority levels. These paragraphs show the following relationships between the final non-pre-emptive region lengths for the tasks at each priority level in priority orderings X_{k-1} and X_k. Paragraph 1 shows that for each priority level l lower than k, $F(l, X_{k-1}) = F(l, X_k)$ and therefore that $\forall l \in lp(k) : F^{\max}(l, X_{k-1}) = F^{\max}(l, X_k)$. Paragraph 2 shows that $F(k, X_{k-1}) \le F(k, X_k)$ and so $F^{\max}(k, X_{k-1}) \le F^{\max}(k, X_k)$. Paragraph 3 shows that $\forall j \in hp(k) \cap lep(i) : F(j, X_{k-1}) \le F(j+1, X_k)$. From paragraph 2, we also know that $F(k, X_{k-1}) \le F(k, X_k)$ and hence $\forall j \in hp(k) \cap lep(i) : F^{\max}(j, X_{k-1}) \le F^{\max}(j, X_k)$. Paragraph 4 shows that $\forall j \in hp(i) : F(j, X_{k-1}) \le F(j, X_k)$ and so we have $\forall j \in hp(i) : F^{\max}(j, X_{k-1}) \le F^{\max}(j, X_k)$. Combining these results for all 4 subsets of priority levels, we have $\forall j : F^{\max}(j, X_{k-1}) \le F^{\max}(j, X_k)$.

In the proof of Theorem 2, n iterations of the priority transformation from $k = n$ to 1, are used to transform the arbitrary priority ordering $X = X_n$ into the priority ordering $X_1 = P$ generated by the FNR-PA algorithm, hence we have $\forall j : F^{\max}(j, P) \le F^{\max}(j, X)$. From (2), $F^{\max}(k, X)$ corresponds to the blocking factor B_{k-1}^{FNR} at priority k-1 in priority order X, due to final non-pre-emptive regions, as the blocking factor at the lowest priority level is always zero □

VI. FNR Length Calculation

The FNR and FNR-PA algorithms presented in the previous section need to compute the *minimum* final non-pre-emptive region length for each task. This can be found using a binary search; however, in this section, we derive an analytical method that can be used instead, thus reducing the overall complexity of the algorithms. We note that the approach presented in [13] cannot be used for this purpose, since it determines the *maximum* final non-pre-emptive region length for each task, and involves computing blocking tolerances which require information about the relative priority ordering of higher priority tasks.

To determine the minimum final non-pre-emptive region length $F(i)$ that ensures the schedulability of the task at priority level i, we effectively need to compute the largest amount of execution that the task can complete pre-emptively without missing its deadline.

As tasks are evaluated lowest priority first, the amount of blocking due to lower priority tasks is known at each step and is given by (2). Further, the set of higher priority tasks is also known, but not their priority order, and so (1) and (3) can be used to compute the priority level-i active period and the number of jobs G_i of a task τ_i that it contains. We then need to compute, for each job g of task τ_i in the priority level-i active-period, the minimum final non-pre-emptive region required to guarantee the schedulability of that job. To do this, we first consider the minimum amount of execution $S_{i,g}(t)$ that task τ_i is certain to be able to perform between the start of its g-th job and some arbitrary time t prior to that job's deadline at $gT_i + D_i$:

$$S_{i,g}(t) = t - B_i - gC_i - \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{t}{T_j} \right\rfloor + 1 \right) C_j \qquad (6)$$

If $S_{i,g}(t) \ge 0$ and the remaining execution time ($C_i - S_{i,g}(t)$) of the job at time t is smaller than the time to its deadline ($gT_i + D_i - t$), then the remaining execution time corresponds to a schedulable final non-pre-emptive region. We need to find the maximum $S_{i,g}(t)$ among all such schedulable time points. The function $S_{i,g}(t)$ is a piece-wise linear function with local maxima corresponding to the release times of higher priority tasks minus one time unit. For each job $g = 0,1,2...G_i - 1$, let $\mathrm{P}_{i,g}$ be the set of time points corresponding to the local maxima of $S_{i,g}(t)$, for $t \in [gT_i, gT_i + D_i - 1]$, including the end of the interval:

$$\mathrm{P}_{i,g} = \left(\forall_{j \in hp(i)} \{hT_j - 1\} \in [gT_i, gT_i + D_i - 1] \right) \cup \{gT_i + D_i - 1\} \qquad (7)$$

To find the maximum $S_{i,g}(t)$ it is sufficient to check the schedulable points in $P_{i,g}$:

$$S_{i,g} = \max_{t \in P_{i,g}} \left\{ S_{i,g}(t) \;\middle|\; C_i - S_{i,g}(t) \le gT_i + D_i - t \wedge S_{i,g}(t) \ge 0 \right\} \qquad (8)$$

Note that the first inequality guarantees that the job has sufficient time to complete its final non-pre-emptive region before its deadline, while the second inequality guarantees that the final non-pre-emptive region can actually start executing at or before time t. If no point in $P_{i,g}$ satisfies both inequalities, then the job is not schedulable, and $S_{i,g}$ is set to a negative value.

The minimum final non-pre-emptive region of the g-th job of task τ_i is given by:

$$F(i,g) = \max(C_i - S_{i,g}, 1) \qquad (9)$$

Taking the maximum over all jobs of task τ_i in the priority level-i-active period determines the minimum final non-pre-emptive region for task τ_i:

$$F(i) = \max_{g=0,1,2...G_i-1} \{F(i,g)\} \qquad (10)$$

Finally, task τ_i is schedulable provided that $F(i) \le C_i$.

This analytical method enables the minimum final non-pre-emptive region for each task to be computed as part of the FNR and FNR-PA algorithms. The run-time complexity of the method is similar to that of a single task schedulability test e.g. the computation required for (5).

The set of valid values for the length of the final non-pre-emptive region of task τ_i may be limited to a subset of values in the range $[1, C_i]$ due to constraints related to nested resource locking or other non-pre-emptive regions. As the schedulability of a task is sustainable with respect to increasing the length of its final non-pre-emptive region (Corollary 1) the minimum valid value can be found by computing the minimum theoretical value as described above, and then selecting the smallest valid value that is no smaller than this computed minimum.

Using the analytical method described in this section, the FNR-PA algorithm effectively requires a maximum of $(n(n+1)/2)$ task schedulability tests to determine an optimal priority and final non-pre-emptive region length assignment, We note that the size of the search space is:

$$n!\prod_{\forall i} C_i$$

Thus, the FNR-PA algorithm represents a significant reduction in complexity, making the problem tractable for the majority of practical applications.

VII. Experimental Evaluation

In this section, we evaluate the performance of the following fixed priority scheduling algorithms:

- FPDS (OPT) with optimal priority and final non-pre-emptive region length assignment using the FNR-PA algorithm introduced in Section V;
- FPDS (DM) with deadline monotonic priority assignment, using the highest-priority-first approach to determining final non-pre-emptive region lengths described in [13];
- FPPS with deadline monotonic priority assignment, which is optimal in this case [31];
- FPNS with optimal priority assignment using Audsley's OPA algorithm [3], [5].
- FPTS Fixed priority Pre-emption Threshold Scheduling [35] using the optimal threshold assignment given in [33] and deadline monotonic priority assignment[4].

A. Parameter generation

The task parameters used in our experiments were randomly generated as follows:

- The UUniFast algorithm [14] was used to generate a set of n utilisation values U_i, with a total utilisation of U.
- Task periods were generated according to a log-uniform distribution[5]. Here the ratio between the maximum and the minimum permissible task period was given by 10^r. By default, this range was 10, i.e. r = 1.
- Task execution times were set based on the utilisation and period selected: $C_i = U_i T_i$.
- The tasks were independent, and hence there were no constraints on the final non-pre-emptive region lengths.
- Task deadlines were either *implicit*, and so equal to their periods, or *constrained* and chosen at random according to a uniform distribution in the range $[C_i + \alpha(T_i - C_i), T_i]$, with α = 0.5 as the default.
- The default taskset cardinality was 10.

In each experiment, the taskset utilisation was varied from 0.03 to 0.99 in steps of 0.03. For each utilisation value, 5000 tasksets were generated and their schedulability determined according to the various scheduling algorithms.

B. Success ratio

In our first set of experiments, we compared the performance of the scheduling algorithms via a metric referred to as the *success ratio*; the proportion of randomly generated tasksets that are schedulable in each case. We also compared the performance of the algorithms to Earliest Deadline First (EDF) scheduling which is an optimal scheduler for fully pre-emptive sporadic tasks with constrained deadlines on a single processor [26].

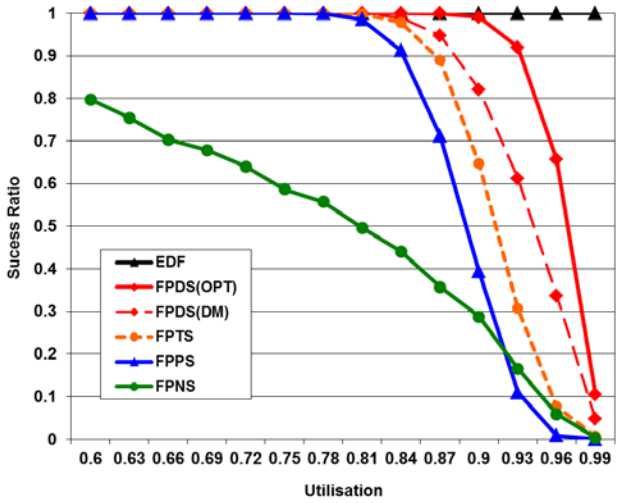

Figure 4: Success ratio for n = 10, D = T

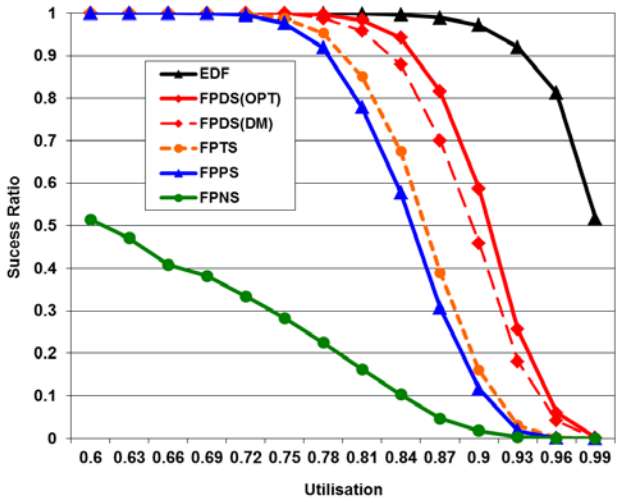

Figure 5: Success ratio for n = 10, D ≤ T

Figure 4 and Figure 5 plot the success ratio for implicit-deadline and constrained-deadline tasksets respectively. These figures show that FPDS significantly improves upon the schedulability obtained using FPPS, FPNS and FPTS. Further, as expected, the optimal approach to FPDS denoted by FPDS(OPT) dominates FPDS(DM) which uses deadline

[4] Deadline monotonic priority assignment is not optimal for FPTS; however, to the best of our knowledge no optimal approach to priority and threshold assignment is known that requires less than exponential time.

[5] The log-uniform distribution of a variable x is such that ln (x) has a uniform distribution.

monotonic priority assignment and calculates final non-pre-emptive region lengths that are optimal only with respect to that priority assignment. Figure 4 clearly illustrates that FPPS and FPNS are incomparable, as the lines for the two algorithms cross. Note the figures are best viewed online in colour.

C. *Weighted schedulability*

In our second set of experiments we compare how the overall performance of each of the scheduling algorithms varies with respect to changes in a specific parameter via a metric referred to as the *weighted schedulability measure* [10]. The following figures show the weighted schedulability measure $Z_y(p)$ for schedulability test y as a function of parameter p. For each value of parameter p, this measure combines results for all of the tasksets generated for all of a set of equally spaced utilisation levels. Let $S_y(\tau, p)$ be the binary result (1 or 0) of schedulability test y for a taskset τ with parameter value p.

$$Z_y(p) = \sum_{\forall \tau} \frac{S_y(\tau).U(\tau)}{U(\tau)} \qquad (11)$$

where $U(\tau)$ is the utilisation of taskset τ. The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2 dimensions [10]. Weighting the individual schedulability results by taskset utilisation reflects the higher value placed on being able to schedule higher utilisation tasksets.

In our weighted schedulability experiments, taskset parameters other than the one that was varied assumed their default values (e.g. n=10, r =1, and α = 0.5).

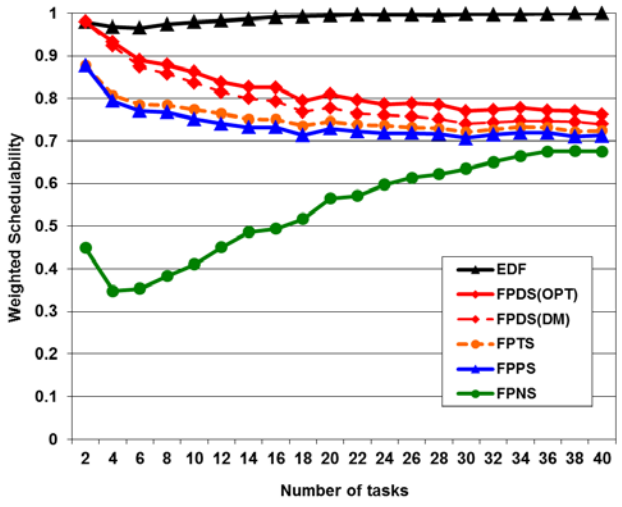

Figure 6: Weighted schedulability as a function of taskset size, D ≤ T

The first parameter we examined was taskset cardinality. Figure 6 shows how the weighted schedulability measure for each of the algorithms varies with increasing taskset size (from 2 to 40 tasks) for tasksets with constrained deadlines.

From Figure 6, we see that FPDS(OPT) significantly outperforms FPNS, FPPS, and FPTS. While the performance of FPDS(DM) is similar to that of FPDS(OPT) for small tasksets, it declines relative to FPDS(OPT) as the number of tasks increases. This happens because with more tasks there is less chance that deadline monotonic priority ordering equates to an optimal priority ordering. With fixed priority pre-emptive scheduling, larger tasksets are in general harder to schedule, hence FPDS, FPPS, and FPTS all show declining performance with increasing taskset size. In contrast, the performance of FPNS improves with increasing taskset size. This is because larger tasksets tend to have tasks with smaller execution times relative to their deadlines, which means that blocking due to non-pre-emptive execution is less detrimental to taskset schedulability.

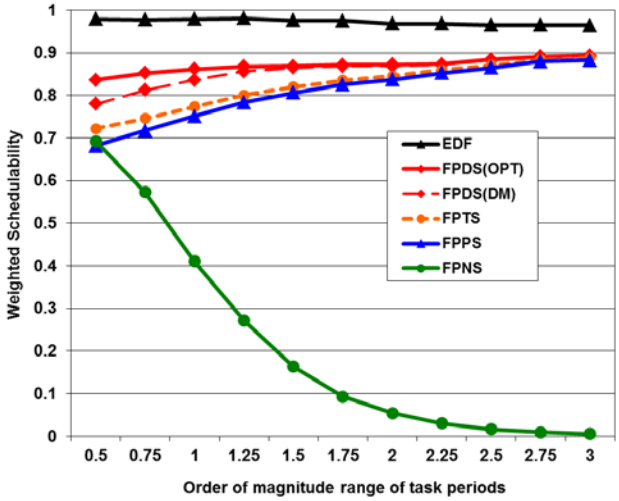

Figure 7: Weighted schedulability as a function of period range, D ≤ T

The second parameter we examined was the range of task periods. Figure 7 shows how the weighted schedulability measure for each of the algorithms varies with the log-range r of task periods given by the ratio 10^r between the maximum and the minimum permissible task period. Here, the value of r was varied from r = 0.5 ($10^{0.5} = 3.16$) to r = 3 ($10^3 = 1000$). Figure 7 shows the results for tasksets with constrained-deadlines.

Figure 7 shows that FPDS(OPT) is particularly effective in scheduling tasksets where the range of task periods is relatively small. This is because with all task periods and deadlines of a similar duration, all of the tasks can typically tolerate a significant amount of blocking and so there is significant scope to choose final non-pre-emptive region lengths that improve schedulability. Further, when tasks have similar periods and deadlines, there is a much greater chance that deadline monotonic priority assignment will not equate to an optimal priority ordering, hence FPDS(OPT) significantly improves upon FPDS(DM) in this case.

As expected, all of the fixed priority pre-emptive scheduling algorithms (FPDS, FPTS, and FPPS) show improved performance as the range of task periods increases. In contrast, FPNS shows rapidly declining performance. This

is because tasks with relatively long periods tend to have large execution times which may be longer than the deadlines of other tasks, making non-pre-emptive scheduling infeasible.

We repeated the weighted schedulability experiments examining taskset cardinality and the range of task periods for tasksets with implicit deadlines. We found that both the results and the conclusions that could be drawn from them were broadly similar to those for constrained-deadline tasksets. The additional figures are omitted for space reasons.

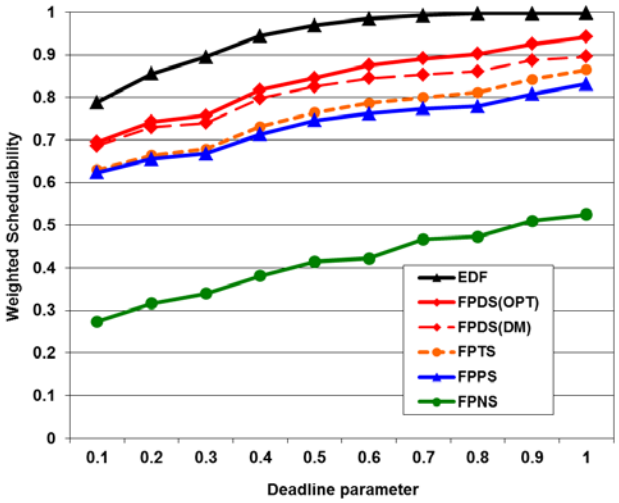

Figure 8: Weighted schedulability as a function of deadline distribution

The third parameter we examined was the range of permissible deadlines relative to the period of each task. Figure 8 shows how the weighted schedulability measure for each of the algorithms varies with this parameter. Here, each task deadline was chosen at random according to a uniform distribution in the range $[C_i + \alpha(T_i - C_i), T_i]$. Figure 8 shows that increasing the range of task deadlines (i.e. smaller α) reduces schedulability in all cases.

We note that even though FPDS has significantly better performance than FPTS in our experiments, no dominance relationship exists between these two scheduling algorithms. There are tasksets that are schedulable with FPTS but not with FPDS and vice-versa, hence the two are incomparable.

VIII. Summary and Conclusions

Fixed priority scheduling with deferred pre-emption (FPDS), dominates both fixed priority fully pre-emptive (FPPS) and fixed priority non-pre-emptive scheduling (FPNS).

The main contribution of this paper is the introduction of an *optimal* algorithm for FPDS. This FNR-PA algorithm is optimal in the sense that it is guaranteed to find a combination of priority assignment and task final non-pre-emptive region lengths that result in a schedulable system under FPDS, whenever such a schedulable combination of these parameters exists. Stated otherwise, the FNR-PA algorithm is able to find a schedulable solution for *any* taskset that is feasible under FPDS. As a consequence of optimising schedulability under FPDS, the FNR-PA algorithm has the notable side-effect that for any given taskset, it minimises the blocking effect due to final non-pre-emptive regions at *every* priority level. Using the analytical method of computing final non-pre-emptive region lengths, derived in Section VI, the FNR-PA algorithm requires at most $(n(n+1)/2)$ task schedulability tests to find an optimal solution, making it tractable for the majority of systems.

We evaluated the performance of optimal FPDS via an experimental investigation. As expected, the experiments verified the dominance of FPDS over FPPS and FPNS. They also showed that the optimal approach, derived in this paper, improves upon previous techniques for FPDS which assumed deadline monotonic priority ordering (DMPO) [13]. We used a simple example with three tasks to illustrate that DMPO is not optimal for FPDS.

A. *Practical applications*

Pre-emptive EDF is the optimal algorithm for scheduling the task model considered in this paper; however, for systems that conform to standards requiring fixed priority scheduling, such as AUTOSAR[6], then FPDS represents an approach that is both efficient to implement and can significantly improve upon the performance of FPPS and FPNS.

The AUTOSAR Operating System standard supports co-operative scheduling of tasks comprising multiple non-pre-emptive regions. In many automotive systems tasks are composed of 50-300 sequential functions [18] each of which could be considered as a non-pre-emptive region. (The use of these non-pre-emptive regions can significantly reduce stack usage, and the implementation of such regions alone can be simpler than using resource locking protocols as there is no requirement to support nesting). The techniques described in this paper are applicable to these systems. In particular, the FNR-PA algorithm introduced in this paper can be used to determine an optimal assignment of task priorities and final non-pre-emptive regions lengths, subject to the constraints imposed on the lengths of such regions by the functions making up each task.

B. *Future work*

The FNR-PA algorithm presented in this paper provides an optimal solution for FPDS for a task model where task execution times are independent of pre-emption and pre-emption costs are negligible. In many real-time systems; however, each pre-emption incurs a cost to do with saving and restoring task contexts, run-queue manipulation and task dispatch. Further, in systems using cache, pre-emption also causes cache-related pre-emption delays (CRPD). Here cache lines evicted by a pre-empting task may need to be reloaded once the pre-empted task has been resumed. For large tasksets, allowing arbitrary pre-emption can result in

[6] AUTOSAR (AUTomotive Open System Architecture) www.autosar.org.

lower priority tasks being pre-empted a large number of times, significantly increasing CRPD to the detriment of schedulability [1], [2]. In fact CRPD can amount to a significant proportion of a task's execution time.

The integration of pre-emption related delays and schedulability analysis is a key area for further research. In future, we plan to integrate schedulability analysis for FPDS with different models of CRPD. Our aim is to obtain optimal or near optimal task priority assignments and non-pre-emptive region allocations that maximise schedulability. For more complex task models this will involve a trade-off between blocking due to non-pre-emptive sections and pre-emption related delays.

ACKNOWLEDGEMENTS

The authors would like to thank Alan Burns for his comments on a draft of this paper. This work was partially funded by the UK EPSRC Tempo project (EP/G055548/1).

REFERENCES

[1] S. Altmeyer, R.I. Davis, C. Maiza "Cache related Pre-emption Delay aware response time analysis for fixed priority pre-emptive systems". In proceedings Real-Time Systems Symposium, pp. 261-271, 2011.

[2] S. Altmeyer, R.I. Davis, C. Maiza "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems" . Real-Time Systems, Volume 48, Issue 5, Pages 499-526, Sept 2012

[3] N.C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Dept. Computer Science, University of York, UK, 1991.

[4] N.C. Audsley, A. Burns, M. Richardson , A.J. Wellings., "Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling". Software Engineering Journal, 8(5), pp 284-292, 1993.

[5] N.C. Audsley, "On priority assignment in fixed priority scheduling", Information Processing Letters, 79(1): 39-44, May 2001.

[6] S.K. Baruah. "The limited-preemption uniprocessor scheduling of sporadic task systems". In Proceedings Euromicro Conference on Real-Time Systems (ECRTS), pp. 137–144, 2005.

[7] S.K. Baruah, A. Burns, "Sustainable Scheduling Analysis". In proceedings Real-Time Systems Symposium, pp. 159-168, 2006.

[8] Baker T.P., "Stack-based Scheduling of Real-Time Processes." *Real-Time Systems Journal* (3)1, pp. 67-100. 1991.

[9] T.P. Baker, S.K. Baruah, "Sustainable multiprocessor scheduling of sporadic task systems". In Proceedings ECRTS, pp. 141-150, 2009.

[10] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability," in *Proceedings of OSPERT*, , pp. 33-44, Brussels, Belgum, 2010.

[11] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, Francesco Esposito, Marco Caccamo. "Preemption points placement for sporadic task sets", In Proceedings Euromicro Conference on Real-Time Systems (ECRTS), Bruxelles, Belgium, June 2010.

[12] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, G. Buttazzo. "Optimal Selection of Preemption Points to Minimize Preemption Overhead", In Proceedings Euromicro Conference on Real-Time Systems (ECRTS), Porto, Portugal, July 2011.

[13] M. Bertogna, G. Buttazzo, G. Yao. "Improving Feasibility of Fixed Priority Tasks using Non-Preemptive Regions", In Proceedings Real-Time Systems Symposium, Vienna, Austria, December 2011.

[14] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. Real-Time Systems, 30(1-2):129–154, 2005.

[15] 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, 42(1-3):63–119, 2009.

[16] R.J. Bril, M.M.H.P. van den Heuvel, U. Keskin, J.J. Lukkien, "Generalized fixed-priority scheduling with limited preemptions", In proceedings Euromicro Conference on Real-Time Systems (ECRTS), pp. 209-220. Pisa, Italy, July 2012.

[17] G.C. Buttazzo, M. Bertogna, G. Yao. "Limited Preemptive Scheduling for Real-Time Systems: A Survey". IEEE Transactions on Industrial Informatics. In press. Downloadable from http://retis.sssup.it/~marko/publi.html

[18] D. Buttle, "Real-Time in the Prime Time" Keynote talk at Euromicro Conference on Real-Time Systems (ECRTS) 2012. Presentation: http://ecrts.eit.uni-kl.de/index.php?id=69.

[19] A. Burns. "Preemptive priority based scheduling: An appropriate engineering approach". S. Son, editor, Advances in Real-Time Systems, pp. 225–248, 1994.

[20] A. Burns, S.K. Baruah "Sustainability in real-time scheduling". Journal of Computing Science and Engineering 2 (1), pp 74-97. 2008.

[21] R.I. Davis, N. Merriam, N.J. Tracey, "How Embedded Applications Using an RTOS can stay within On-chip Memory Limits". In proceedings Work in Progress and Industrial Experience Sessions, Euromicro Conference on Real-Time Systems (ECRTS), 2000.

[22] R.I. Davis, A. Burns. "Robust Priority Assignment for Fixed Priority Real-Time Systems". In proceedings Real-Time Systems Symposium, pp. 3-14. 2007

[23] R.I. Davis, A. Burns. "Response Time Upper Bounds for Fixed Priority Real-Time Systems". In proceedings Real-Time Systems Symposium, Barcelona, Spain, 2008.

[24] R.I. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems". IEEE Transactions on Computers, (Vol. 57, No. 9) pp. 1261-1276, September 2008.

[25] R.I. Davis and A. Burns "Robust priority assignment for messages on Controller Area Network (CAN)". Real-Time Systems, Volume 41, Issue 2, pages 152-180, February 2009.

[26] M.L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes". In Proceedings of the IFIP congress, pages 807-813, 1974.

[27] L. George, N. Rivierre, M. Spuri, "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", INRIA Research Report, No. 2966, September 1996.

[28] M. Joseph, P.K. Pandya, "Finding Response Times in a Real-time System". The Computer Journal, 29(5), pages 390–395, 1986.

[29] J.P. Lehoczky, L. Sha, Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour". In Proceedings Real-Time Systems Symposium, pp. 166–171, 1989.

[30] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In Proceedings Real-Time Systems Symposium, pp. 201–209, 1990.

[31] J.Y.-T. Leung, J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks". Performance Evaluation, 2(4), pp. 237-250, 1982.

[32] C.L. Liu, J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1) pp. 46-61, 1973.

[33] M. Saksena and Y. Wang. "Scalable real-time system design using preemption thresholds". In Proceedings Real-Time Systems Symposium, 2000.

[34] K.W.Tindell, A. Burns, A.J.Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks". Real-Time Systems. Volume 6, Number 2, pp. 133-151, 1994.

[35] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with pre-emption threshold. In Proceedings RTCSA'99, Hong Kong, China, December 13-15, 1999.

[36] G. Yao, G. Buttazzo, M. Bertogna. "Bounding the Maximum Length of Non-Preemptive Regions Under Fixed Priority Scheduling", In Proceedings RTCSA 2009, Beijing, China, August 2009.

[37] G. Yao, G. Buttazzo, M. Bertogna. "Feasibility Analysis under Fixed Priority Scheduling with Fixed Preemption Points", In Proceedings RTCSA 2010, Macau, China, August 2010.