Optimal Priority and Threshold Assignment for Fixed-priority Preemption Threshold Scheduling

Leo Hatvani
Technische Universiteit Eindhoven (TU/e)
The Netherlands
l.hatvani@tue.nl

Sara Afshar
Mälardalen University (MDH)
Västerås, Sweden
sara.afshar@mdh.se

Reinder J. Bril
Technische Universiteit Eindhoven (TU/e)
The Netherlands
Mälardalen University (MDH)
Västerås, Sweden
r.j.bril@tue.nl

ABSTRACT

Fixed-priority preemption-threshold scheduling (FPTS) is a generalization of fixed-priority preemptive scheduling (FPPS) and fixed-priority non-preemptive scheduling (FPNS). Since FPPS and FPNS are incomparable in terms of potential schedulability, FPTS has the advantage that it can schedule any task set schedulable by FPPS or FPNS and some that are not schedulable by either. FPTS is based on the idea that each task is assigned a priority and a preemption threshold. While tasks are admitted into the system according to their priorities, they can only be preempted by tasks that have priority higher than the preemption threshold.

This paper presents a new optimal priority and preemption threshold assignment (OPTA) algorithm for FPTS which in general outperforms the existing algorithms in terms of the size of the explored state-space and the total number of worst case response time calculations performed. The algorithm is based on back-tracking, i.e. it traverses the space of potential priorities and preemption thresholds, while pruning infeasible paths, and returns the first assignment deemed schedulable.

We present the evaluation results where we compare the complexity of the new algorithm with the existing one. We show that the new algorithm significantly reduces the time needed to find a solution. Through a comparative evaluation, we show the improvements that can be achieved in terms of schedulability ratio by our OPTA compared to a deadline monotonic priority assignment.

CCS Concepts

•Computer systems organization → Embedded software; •Software and its engineering → Real-time schedulability;

1. INTRODUCTION

Scheduling theory has been widely studied over the years to provide effective scheduling solutions for real-time embedded systems. Scheduling algorithms can be divided into fixed-priority and dynamic-priority, where the priorities of tasks stay the same throughout the execution of the system for the former ones and change depending on some parameter for the latter ones. Even though it can be said that they lack the flexibility of dynamic-priority algorithms, fixed-priority scheduling algorithms are a de facto standard in industry due to their lower implementation and execution overhead compared to dynamic-priority algorithms.

Another way of classifying scheduling algorithms is whether they allow a task that is currently executing to be preempted or not, these are preemptive and non-preemptive scheduling algorithms. Recently, new scheduling approaches have been proposed as an alternative to the extremes of fully preemptive and non-preemptive scheduling. One such scheduling approach is called *fixed-priority preemption threshold scheduling* (FPTS) [13, 12] where each task τ_i is annotated by a preemption threshold θ_i besides its priority π_i. The preemption threshold for a task specifies which tasks can preempt it and is always greater than or equal to the priority of that task. If and only if a task's priority is higher than the preemption threshold of the currently executing task, it is allowed to preempt the executing task. This means that as soon as the task starts executing, its effective priority is increased to its preemption threshold.

There are two special cases of the FPTS scheduling algorithm. First, when the preemption threshold of every task is equal to the task's priority, FPTS becomes equivalent to the FPPS scheduling algorithm. Second, when all preemption thresholds are equal to the highest priority in the system, FPTS becomes equivalent to FPNS, as no task can preempt any other task.

FPTS improves the schedulability compared to both approaches by leveraging the preemption of higher priority tasks under FPPS and the blocking incurred by non-preemptive execution of lower priority tasks under FPNS. An *optimal priority and threshold assignment* (OPTA) algorithm for FPTS is any algorithm that for a given set of tasks determines if it can be scheduled by means of FPTS and when it can be scheduled provides task priorities and thresholds. The original OPTA for FPTS has been introduced by Wang and Saksena [13]. However, the high computation time required to find a feasible solution using this algorithm, if such exists, warrants research into more efficient approaches [6]. In this paper, we introduce a new OPTA algorithm for FPTS which effectively can decrease the computation time of finding a feasible solution. Our algorithm is based on (i) backtracking, (ii) pruning to reduce the search space, and (iii) a heuristic to traverse the search space.

Contributions: In this paper we propose an algorithm for optimal priority and threshold assignment. Moreover, we show by means of experimental results that the computation time for finding a priority and threshold assignment for a set

EWiLi'16, October 6th, 2016, Pittsburgh, USA.

of tasks that can make them schedulable, if such a combination exists, is growing considerably slower compared to the existing solution when the number of tasks increases. Further, we present the schedulability results of comparing our proposed algorithm with FPPS and FPTS under a deadline monotonic assignment. Finally, we present a comparative evaluation of FPTS with a deadline monotonic priority assignment and optimal preemption thresholds and our OPTA, which clearly illustrates the advantage of our OPTA.

2. MOTIVATION EXAMPLE

In this section we show by means of an example that the deadline monotonic priority assignment is not optimal for FPTS.

Table 1: Specifications of $\mathcal{T}_1$ task set.

$task$	C_i	$T_i = D_i$	π_i^{DM}	π_i	θ_i	R_i
τ_1	1	7	1	1	1	1
τ_2	8	23	2	2	2	21
τ_3	10	25	3	4	2	25
τ_4	3	33	4	3	2	25

Let us consider the task set $\mathcal{T}_1$ which consists of 4 tasks as presented in Table 1. The tasks are characterized by their worst-case computation time C_i, minimum inter-arrival time T_i, deadline D_i, preemption thresholds θ_i, and priority π_i. The smaller numbers denote higher priorities. All tasks have implicit deadlines, i.e. $T_i = D_i$, and the priorities $\pi_i^{\mathrm{DM}}$are assigned based on deadline monotonic priority assignment, i.e. the task with the smallest deadline has the highest priority. At utilization $U^{\mathcal{T}_1} \approx 0.982$, $\mathcal{T}_1$ is not schedulable under FPTS. If the priorities of τ_3 and τ_4 are exchanged (as in π_i) $\mathcal{T}_1$ becomes schedulable under FPTS with preemption thresholds θ_i and worst-case response times R_i.

3. RELATED WORK

Fixed Preemption Points (FPP) scheduling has been proposed by Burns [4]. Under FPP, preemption is allowed only at predefined points during a task's execution. As a results, a task is divided into a number of non-preemptive execution chunks. The preemption of a higher priority task is postponed until the next preemption point of a running task. This approach is also referred to as cooperative scheduling.

Preemption Threshold Scheduling (PTS) has been proposed by Wang and Saksena in [13]. Under this approach, besides a regular priority, each task is assigned a preemption threshold up to where it can prevent preemptions. The preemption is allowed only when the priority of a ready task is higher than the threshold of the running task.

Worst-case response time analysis for FPTS as well as an optimal priority and preemption threshold assignment algorithm was first presented in [13]. Later, in [11], it has been shown that the analysis in [13] was optimistic, and a revised analysis was presented. Exact analysis has been presented in [8].

Deferred Preemptions Scheduling (DPS) has first been introduced by Baruah [1] under Earliest Deadline First (EDF). Under this approach, for each task the longest interval that can be executed non-preemptively is specified. Two types of implementation exist for this approach based on how the non-preemptive regions are implemented: Floating model and Activation-triggered model implementation. Under the former model, non-preemptive regions are specified in the code by inserting specific primitives that disable and enable preemption. Under the latter model, non-preemptive regions are specified by setting a timer at the arrival of a higher priority task which lasts for the specified non-preemptive interval.

Worst-case response time analysis of periodic real-time tasks under fixed-priority scheduling with deferred preemption (FPDS) has been studied in [4, 5, 9]. In [3], Bril et. al have shown that the analysis presented in [4, 5] is both pessimistic and optimistic where they have revised the analysis.

4. SYSTEM MODEL

4.1 Task Specification

The system consists of a single processor including a set of n independent sporadic tasks. Each task τ_i is denoted by $< C_i, D_i, T_i >$, where $C_i \in \mathbb{R}^+$ denotes the worst-case execution time, $D_i \in \mathbb{R}^+$ denotes the relative deadline and $T_i \in \mathbb{R}^+$ is the minimum inter-arrival time. We assume arbitrary deadlines, i.e., $D_i \leq T_i$, or $D_i > T_i$.

4.2 FPTS Scheduling

Tasks are further annotated by unique priorities $\pi_i \in \mathbb{N}$ and preemption thresholds $\theta_i \in \mathbb{N}$. For both, smaller values denote higher priorities. Whenever we compare priorities or thresholds, we compare their numerical values, i.e. $\pi_i < \pi_j$ means that task π_i has a higher priority than π_j. Preemption thresholds are always greater than or equal to the corresponding priorities $\theta_i \leq \pi_i$.

Given these priorities and thresholds, the tasks are scheduled as follows. If there are no running tasks, the task τ_i with the highest priority among the simultaneously released tasks starts executing. As soon as it starts executing it can be preempted only by tasks τ_j that have a priority greater than its preemption threshold $\pi_j < \theta_i$. In the situation when the executing task has a higher priority than the preempted task τ_i and waiting task τ_j where $\theta_i \leq \pi_j$, τ_i will always start executing before τ_j.

5. RESPONSE TIME ANALYSIS

5.1 Analysis

While worst-case response time analysis for FPTS policy has been published several times [13, 11], in this paper we employ the exact analysis presented by Keskin et al. [8] with the added assumption that there is no jitter.

A task τ_i releases an infinite series of jobs τ_{ik}. Each of these jobs faces two kinds of delays to its execution: blocking and interference. Blocking (1) comes from the lower priority tasks that have a preemption threshold equal to or higher than the priority of the current task.

$$B_i = \max(0, \max_{\forall j:\theta_j \leq \pi_i < \pi_j} C_j) \tag{1}$$

Since the sequence of jobs released by the task τ_i is infinite, we need to determine how many jobs should be analyzed to determine whether the entire task is schedulable or not. This calculation is based on the *worst-case level-i active period*

[3] and is given as the smallest positive L_i that satisfies (2).

$$L_i = B_i + \sum_{\forall j:\pi_j \leq \pi_i} \left\lceil \frac{L_i}{T_j} \right\rceil C_j \tag{2}$$

The maximum number l_i of jobs of τ_i that can be executed in this level-i active period is given by (3).

$$l_i = \left\lceil \frac{L_i}{T_i} \right\rceil \tag{3}$$

Given the knowledge of how many jobs need to be analyzed, we can compute their worst-case start time S_{ik} for $0 \leq k < l_i$ as the smallest positive value S_{ik} that satisfies (4).

$$S_{ik} = \begin{cases} B_i + kC_i + \sum_{\forall j:\pi_j<\pi_i} \left\lceil \frac{S_{ik}}{T_j} \right\rceil C_j & \text{if } B_i > 0 \\ kC_i + \sum_{\forall j:\pi_j<\pi_i} \left(1 + \left\lfloor \frac{S_{ik}}{T_j} \right\rfloor\right) C_j & \text{if } B_i = 0 \end{cases} \tag{4}$$

Likewise, other tasks may preempt the observed job, thus causing interference. This quantity is encompassed by calculating each job's worst-case finalization time F_{ik}. It can be found as the smallest positive F_{ik} that satisfies (5).

$$F_{ik} = \begin{cases} S_{ik} + C_i + \\ \sum_{\forall j:\pi_j<\theta_i} \left(\left\lceil \frac{F_{ik}}{T_j} \right\rceil - \left\lceil \frac{S_{ik}}{T_j} \right\rceil\right) C_j & \text{if } B_i > 0 \\ S_{ik} + C_i + \\ \sum_{\forall j:\pi_j<\theta_i} \left(\left\lceil \frac{F_{ik}}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_{ik}}{T_j} \right\rfloor\right)\right) C_j & \text{if } B_i = 0 \end{cases} \tag{5}$$

Finally, according to (6), we compute the worst-case response time as the maximum of response times of all jobs within the worst-case level-i active period.

$$R_i = \max_{\forall k:0\leq k<l_i} (F_{ik} - kT_i) \tag{6}$$

5.2 Definition and Observations

First, let us define the key concept that is the basis of our algorithm. *Blocking tolerance* for tasks with non-preemptive regions was introduced by Lortz and Shin [10] and later exploited by Yao et al. [14]. The same concept is utilized in this work with the specialization that the entire task is considered either preemptive or non-preemptive depending on the relevant priorities and thresholds.

Definition 1 (Blocking tolerance). *Given a schedulable task τ_i in the context of a set of tasks $\mathcal{T}$, its blocking tolerance β_i is the maximum amount of blocking that it can experience from a lower priority task while staying schedulable.* □

In other words, blocking tolerance is the maximum computation time that can be assigned to a task of a lower priority that blocks the observed task without compromising the observed task's schedulability. Next, let us derive two observations that will be utilized in our algorithm.

Lemma 1. *Given two tasks τ_i and τ_j with $C_j > \beta_i$, task τ_i will be rendered unschedulable if task τ_j blocks it ($\pi_j > \pi_i \wedge \theta_j \leq \pi_i$) or is assigned a priority greater than π_i.* □

Based on Definition 1, the first part of Lemma 1 is immediately true. To observe that the second part is also true, we can inspect that while blocking value B_i in (4) influences only the start time of the task, if the same computation time is allocated to a higher priority, it is introduced into the computation of starting time (4) and optionally finishing time (5) as C_j.

Corollary 1. *Let τ_i be a task and $\mathcal{T}$ be a set of schedulable tasks with assigned priorities and thresholds. If task τ_i is not schedulable in the context of $\mathcal{T}$ with the lowest priority $\forall \pi_j \in \mathcal{T} : \pi_i > \pi_j$ and any preemption threshold, it will not be schedulable if additional tasks are added to $\mathcal{T}$.* □

From the worst case response time equations, it is trivial to deduce that adding more interference will never reduce the response time of a task.

6. OPTIMAL PRIORITY AND THRESHOLD ASSIGNMENT

In this section, we present an OPTA algorithm for FPTS. The algorithm is optimal in the sense that it will find a feasible priority and thresholds assignment that makes a task set schedulable if such solution exists. Similar to the OPTA in the work by Wang and Saksena [13] (including the correction discussed in Section 7), the algorithm proposed in this paper is based on (i) backtracking, (ii) pruning the search space, and (iii) a heuristic for traversing the search space. However, unlike the Wang-Saksena OPTA, our algorithm (1) assigns priorities in a descending order, i.e. from the highest to the lowest priority, rather than in ascending order, (2) determines preemption thresholds of tasks while the priority ordering is determined, rather than determining preemption thresholds after the priority ordering is completed, and (3) uses the notion of blocking tolerance rather than lateness in the heuristic.

6.1 State-space Pruning

FPTS-OPTA algorithm uses three types of pruning for infeasible priority orderings to reduce the searched state-space. All three types of pruning utilize the same information about blocking tolerance computed once per recursive call for every unassigned task.

First, any partial priority ordering that results in a blocking tolerance less than 0, for a yet unassigned task, is immediately rejected (lines 6–8 of Algorithm 1). This pruning method follows from Corollary 1.

Second, if there are two unassigned tasks τ_i, and τ_j that have computation times greater than the blocking tolerance of the other task ($C_i > \beta_j$, and $C_j > \beta_i$), then there is no sequence of priority orderings that can make both tasks schedulable. Therefore we reject the current partial priority ordering (lines 9–11). This is the immediate consequence of Lemma 1. Even if these two tasks do not experience blocking from each other, the one at a higher priority will be introducing interference.

And third, let there be two unassigned tasks τ_i, and τ_j where the blocking tolerance β_i of τ_i is smaller than the computation time C_j of the other task τ_j ($\beta_i < C_j$) and blocking tolerance of τ_j is larger than the computation time of the task τ_i ($\beta_j \geq C_i$). Then there is only one sequence of priorities under which these tasks can be made schedulable: τ_i at a higher priority than τ_j. Therefore, it is safe to remove the task τ_j from the current priority (lines 12–19).

Algorithm 1 FPTS-OPTA

Input:
A set of tasks $\mathcal{T}$, and $\{C_i, T_i, D_i\}$ $\forall \tau_i \in \mathcal{T}$.
$\mathcal{H} = \emptyset$ ▷ The set of scheduled tasks.
$\mathcal{L} = \mathcal{T}$ ▷ The set of tasks to be scheduled.
$Prio = 1$ ▷ The highest priority.
Output:
The schedulability of the task set, π_i, and θ_i $\forall \tau_i \in \mathcal{T}$.

```
1: function SEARCH(H, L, Prio)
2:     if L = ∅ then
3:         return SCHEDULABLE
4:     end if
5:     ∀τ_i ∈ L : β_i, θ'_i ← CALCULATEBT(H, τ_i, Prio);
6:     if (∃τ_i ∈ L : β_i < 0) then
7:         return UNSCHEDULABLE
8:     end if
9:     if (∃τ_i, τ_j ∈ L : i ≠ j ∧ β_i < C_j ∧ β_j < C_i) then
10:        return UNSCHEDULABLE
11:    end if
   ▷ L' is a list of tasks that can be assigned priority Prio.
12:    L' ← L
13:    for (τ_i ∈ L) do
14:        for (τ_j ∈ L \ {τ_i}) do
15:            if (β_i < C_j ∧ β_j ≥ C_i) then
16:                L' ← L' \ {τ_j}
17:            end if
18:        end for
19:    end for
20:    L' ← SORTβINC(L', β⃗)
21:    for (τ_i ∈ L') do
22:        H ← H ∪ {τ_i}
23:        L ← L \ {τ_i}
24:        π_i ← Prio
25:        θ_i ← θ'_i
26:        if (SEARCH(H,L,Prio+1)=SCHEDULABLE) then
27:            return SCHEDULABLE
28:        end if
29:        H ← H \ {τ_i}
30:        L ← L ∪ {τ_i}
31:    end for
32:    return UNSCHEDULABLE
33: end function
```

6.2 State-space Traversal Heuristic

The idea behind assigning priorities in a descending order is that, if a task is at a lower priority, it will experience a higher amount of interference which in turn leads to a smaller tolerance for blocking.

On the other hand, smaller blocking tolerance reduces the opportunity for lower priority tasks (which are not schedulable by their current priority level) to increase their preemption threshold for a try to become schedulable. To exploit these properties, we employ a heuristic for traversal of the potential priority assignment state-space. We first allocate those tasks to the highest priorities that have the lowest blocking tolerance.

6.3 FPTS-OPTA Algorithm

The algorithm divides the task set $\mathcal{T}$ into two groups: (i) a set of higher priority tasks $\mathcal{H}$ which have their priorities and thresholds assigned (ii), and a set of lower priority tasks $\mathcal{L}$ with unassigned priorities and thresholds.

The algorithm starts with an empty set $\mathcal{H}$, and all tasks to be processed in $\mathcal{L}$. Additionally, a parameter of which priority should be processed is passed to the algorithm, which is initially the highest available priority in the system.

The algorithm finds a solution if no task remains in $\mathcal{L}$ (lines 2–4 in the algorithm). In each iteration of the algorithm, first the blocking tolerance and preemption thresholds of all tasks in $\mathcal{L}$ are calculated (line 5 in the algorithm). The preemption threshold θ'_i is the highest one with which all tasks in $\mathcal{H}$ can be scheduled if the task were added to the set at the priority $Prio$. This preemption threshold is subsequently used to calculate blocking tolerance β_i of the same task.

There are two pruning conditions to determine if the task set is unschedulable: In a given priority level, if (i) the blocking tolerance of any task is a negative value (lines 6–8 in the algorithm), and (ii) there exist two tasks where blocking tolerance of each is lower than the execution time of the other, i.e., $(\exists \tau_i, \tau_j \in \mathcal{L} : i \neq j \wedge \beta_i < C_j \wedge \beta_j < C_i)$ (lines 9–11 in the algorithm).

Another condition for pruning the tasks from a specific priority level is that, the execution time of those tasks cannot satisfy the blocking tolerance of some tasks at that priority level, i.e., $(\exists \tau_i, \tau_j \in \mathcal{L} : i \neq j \wedge \beta_i < C_j \wedge \beta_j \geq C_i)$ (lines 12–19 in the algorithm) which means that τ_i is not schedulable at a lower priority level, thus τ_j is pruned from the current priority level. The tasks that should be explored at the current priority level are added to list $\mathcal{L}'$. Tasks in $\mathcal{L}'$ are then sorted in ascending order of their blocking tolerances by SORTβINC function (line 20 in the algorithm).

The task τ_i at the head of $\mathcal{L}'$, i.e., the task with minimum blocking tolerance, is selected to be assigned to the current priority level, removed from $\mathcal{L}$, and added to $\mathcal{H}$ (lines 22 to 25 in the algorithm).

The algorithm subsequently goes one step deeper in the recursion for the next (lower) priority level (line 26 in the algorithm). If the recursion returns a SCHEDULABLE value, the value is returned, otherwise the algorithm proceeds to try to schedule the next task from the list $\mathcal{L}'$.

7. FIXING WANG-SAKSENA ALGORITHM

A prerequisite for a comparative evaluation of our algorithm and the algorithm presented by Wang and Saksena [13] was to implement both algorithms. During the implementation of the algorithm of Wang and Saksena, we noticed that it does not explore the entire state-space.

After inspecting the pseudo-code in Figure 3 of [13], we have determined that the most probable original intention for the algorithm was to continue the state space exploration if an unschedulable configuration was found. A simple removal of line 4 from the pseudo-code results in an algorithm that completely explores the state-space of the potential priority orderings. Further, this closely reflects the approach used in the same algorithm on lines 22–24.

8. EVALUATION

In this section we present the results of our experiments. As the first set of experiments, we have measured the complexity of our proposed algorithm compared to Wang and Saksena's algorithm[1] [13]. Since both algorithms execute in exponential time, we chose two different measures to compare them. The number of recursive calls to the main function

[1]After applying the corrections outlined in Section 7.

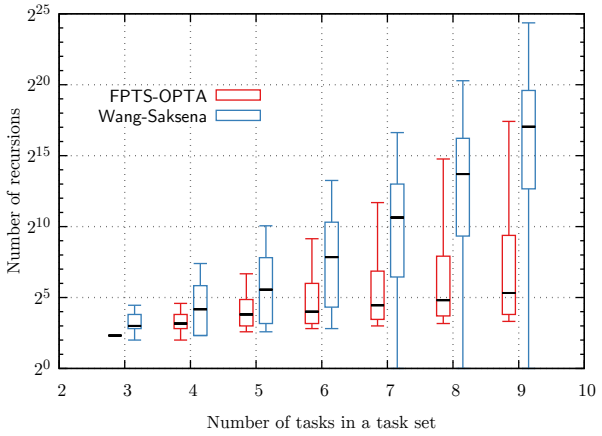

Figure 1: Number of recursions for varied task set cardinality.

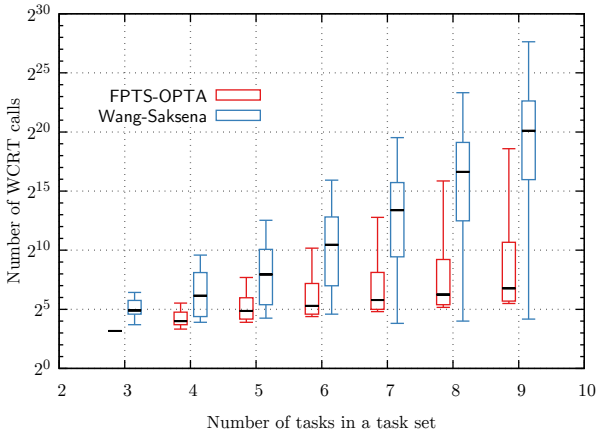

Figure 2: Number of WCRT calls for varied task set cardinality.

that corresponds to the number of explored partial priority orderings. And the number of worst-case response time (WCRT) computation calls.

For the second set of experiments, we compared the schedulability ratios of task sets under (*i*) FPPS with deadline monotonic priorities, (*ii*) FPTS with deadline monotonic priorities and optimal preemption thresholds (FPTS-DM), and (*iii*) FPTS with our OPTA (FPTS-OPTA). These comparisons were done for constrained deadlines ($D_i \leq T_i$), and deadlines greater than periods ($D_i > T_i$).

8.1 Experimental Setup

For every experiment, 5000 task sets were randomly generated using UUnifast algorithm [2] for every x-axis point. Task's inter-arrival times were randomly chosen from the range $[10, 1000]$. Task's deadlines were randomly selected from the range $[C_i + \alpha(T_i - C_i), T_i], 0 < \alpha \leq 1$ for the constrained deadlines, and $[T_i, C_i + \alpha(T_i - C_i)], \alpha > 1$ for deadlines greater than periods, where α is a deadline scaling factor. As a special case $\alpha = 1$ corresponds to implicit deadlines ($D_i = T_i$).

In our experiments, the following default parameters were used (except where otherwise noted): implicit deadlines ($\alpha = 1$), the task set utilization set to 0.9, and the task set cardinality set to 8.

8.2 Complexity Results

While Wang-Saksena algorithm is entirely based on WCRT computation, our algorithm utilizes blocking tolerance computation. To achieve a common ground between the algorithms, we have implemented blocking tolerance and maximum preemption threshold computation using WCRT calls. Furthermore, we have compared the algorithm efficiency only for the task sets that are not FPPS deadline monotonic schedulable.

The number of calls to the main function and number of calls to WCRT computing function were measured when the task set cardinality varies from 3 to 9 in increments of 1. The results are shown by means of box plots (a.k.a. box and whisker diagram) in Figures 1 and 2. Each graph represents the data using five values: (1) the minimum value as the lowest line in the graph, (2) the first quartile value as the lower edge of the rectangle, (3) the median value represented by the black line within the rectangle, (4) the third quartile value as the upper edge, and (5) the maximum value as the top line.

From Figures 1 and 2, we can observe that the number of WCRT calls as well as main function calls are orders of magnitude lower under our optimal algorithm compared to Wang-Saksena algorithm. This correlates to a significant improvement in execution time to determine the existence of a solution. Further, it can be seen that the growth of the measured median values in the graphs is steeper under Wang-Saksena algorithm compared to FPTS-OPTA algorithm. It is interesting to observe that the maximum number of recursions under FPTS-OPTA algorithm is smaller than the third quartile for the Wang-Saksena algorithm (Figure 1). And the maximum number of WCRT calls for FPTS-OPTA is less than the median value for the Wang-Saksena algorithm (Figure 2). However, the results also show that there are some task sets for which FPTS-OPTA does not outperform Wang-Saksena algorithm.

8.3 Schedulability Results

In the second set of experiments, we have first analyzed task sets with implicit deadlines with varied utilization and task set cardinality, then we have analyzed the impact of reducing or increasing deadlines relative to periods. Finally, we have analyzed the schedulability for varied task set cardinality when deadlines are greater than periods.

8.3.1 Implicit Deadlines

The results of analyzing schedulability ratio of the three different priority assignment algorithms, FPPS, FPTS-DM, and FPTS-OPTA for implicit deadlines, 0.9 task set utilization, and varied number of tasks can be seen in Figure 3. From the results, we can see that the optimal priority and preemption thresholds assignment algorithm has no significant reduction in schedulability ratio for the increasing number of tasks whereas FPTS-DM and FPTS have significant reductions for every added task. The results for 8 tasks, and increasing utilization from 0.6 to 0.95 in 0.025 increments are shown in Figure 5. From it, we can observe up to 20% increase in schedulability when the optimal priority and preemption threshold assignment was used.

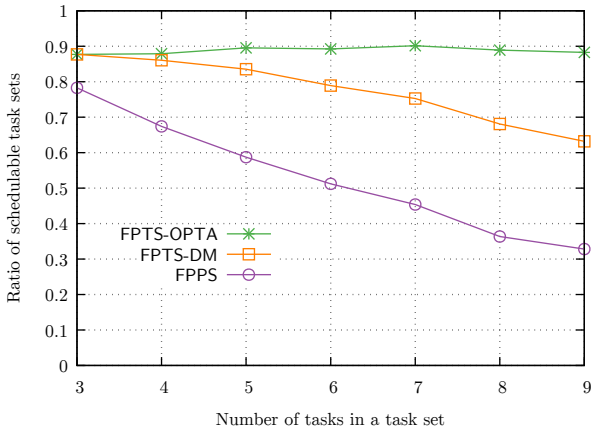

Figure 3: Task set schedulability ratio for varying task set cardinality with implicit deadlines.

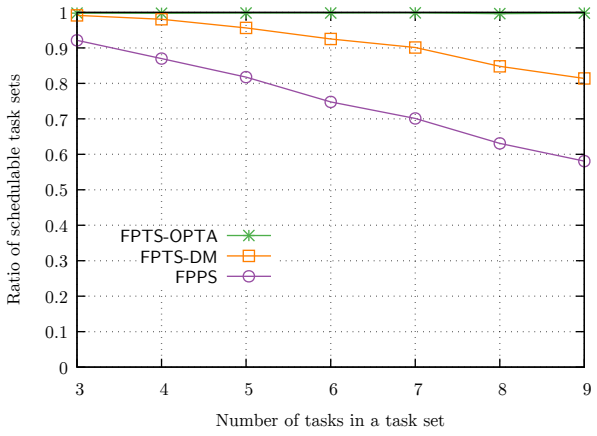

Figure 4: Task set schedulability ratio for varying task set cardinality with $\alpha = 1.5$.

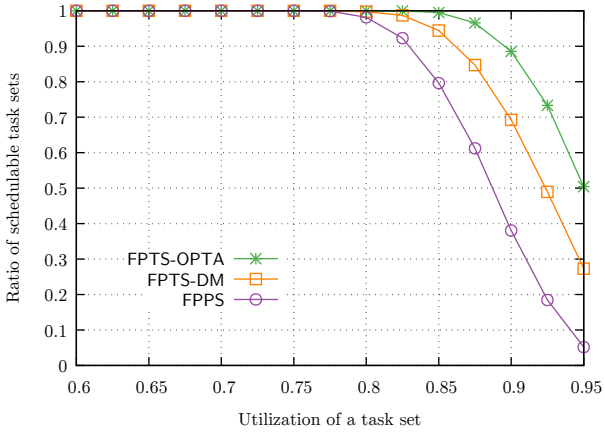

Figure 5: Task set schedulability ratio for varying task set utilization with implicit deadlines.

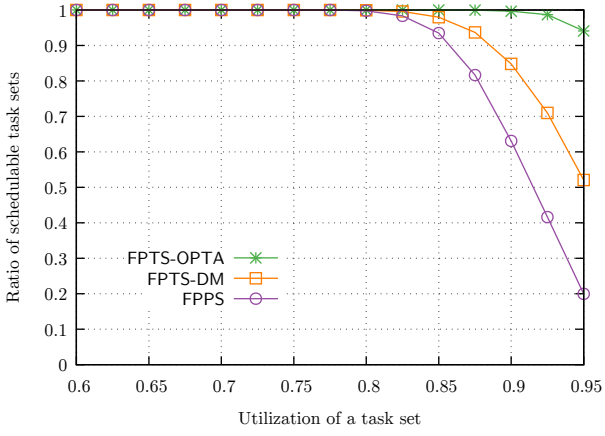

Figure 6: Task set schedulability ratio for varying task set utilization with $\alpha = 1.5$.

8.3.2 *Scaled Deadlines*

To understand the impact of various deadline to period ratios, we have analyzed schedulability ratios for varying values of deadline scaling factor α from 0.1 to 3. The task set utilization was set to 0.9 and 8 tasks per task set were considered.

With constrained deadlines ($D_i \leq T_i$, and $0 < \alpha \leq 1$), depicted in Figure 7, reducing α reduces the schedulability and the difference between the different algorithms. For deadlines larger than periods ($D_i > T_i$, and $\alpha > 1$), as shown in Figure 8, an increase in α increases schedulability for all three algorithms and reduces the difference between the algorithms. With our setup, we observed the largest difference between the FPTS-OPTA and FPTS-DM for $\alpha = 1.1$ and is slightly above 20%.

8.3.3 *Deadlines Greater than Periods*

Finally, we have conducted the original two experiments for varied task set cardinality and utilization with deadlines greater than periods using $\alpha = 1.5$. The results are shown in Figures 4 and 6. In both cases, the effect of increasing α beyond 1 increases schedulability and reduces the gap between the different algorithms.

9. CONCLUSION

In this paper we proposed a new optimal priority and preemption threshold assignment algorithm for FPTS.

The algorithm is based on backtracking, pruning the search space and a heuristic for traversing the search space. By generating a large number of synthetic task sets, we have compared our algorithm with the original algorithm by Wang and Saksena. The results show that our algorithm clearly outperforms the Wang-Saksena algorithm in terms of number of recursive calls to the main function of each algorithm and number of worst-case response time calculations required. Even though, special cases exist for which Wang-Saksena performs better than our algorithm.

Further, we have used our algorithm to evaluate the relevance of using a complex algorithm to determine the priority assignment compared to a simple deadline monotonic assignment. With up to 20% observed increase in schedulability, we can conclude that it is worth having the option of optimal priority assignment for cases that cannot be scheduled by means of deadline monotonic priorities and optimal pre-

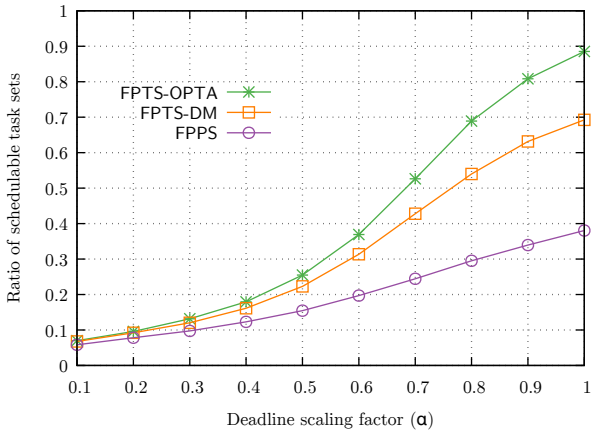

Figure 7: Task set schedulability ratio for varying values of α and constrained deadlines.

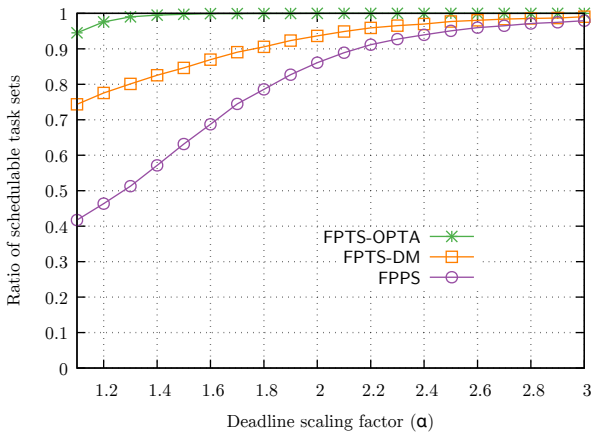

Figure 8: Task set schedulability ratio for varying values of α and deadlines greater than periods.

emption thresholds. However, since the algorithm still executes in exponential time, it may be impractical to execute it for very large task sets.

A specific case that we are looking forward to investigating is the application of the OPTA algorithm to reduce the schedulability difference between FPTS and the restricted variant of FPTS implemented using native non-preemptive groups on an AUTOSAR/OSEK[2] compatible platform [7]. In this restricted implementation, only a subset of potential preemption threshold configurations are viable and thus schedulability is reduced.

The problem itself, of whether it is possible to construct an efficient algorithm for optimal priority and preemption threshold assignment for FPTS, still remains open.

ACKNOWLEDGMENTS

This work is supported by the ARTEMIS Joint Undertaking project EMC[2] (grant agreement 621429).

[2]AUTOSAR/OSEK standard can be found at http://www.autosar.org/

10. REFERENCES

[1] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In 17^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 137–144, Jul. 2005.

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

[3] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In 19^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 269–279, Jul. 2007.

[4] A. Burns. Advances in real-time systems. chapter Preemptive Priority-based Scheduling: An Appropriate Engineering Approach, pages 225–248. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[5] A. Burns and A. J. Wellings. Restricted tasking models. In 8^{th} *International Workshop on Real-Time Ada (IRTAW)*, pages 27–32, New York, NY, USA, 1997. ACM.

[6] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65:64 – 82, 2016.

[7] L. Hatvani and R. J. Bril. Schedulability using native non-preemptive groups on an AUTOSAR/OSEK platform. In 20^{th} *Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sep. 2015.

[8] U. Keskin, R. J. Bril, and J. J. Lukkien. Exact response-time analysis for fixed-priority preemption-threshold scheduling. In 15^{th} *IEEE Conf. on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, Sep. 2010.

[9] S. Lee, C.-G. Lee, M. Lee, S. Min, and C. Kim. Limited preemptible scheduling to embrace cache memory in real-time systems. In F. Mueller and A. Bestavros, editors, *Languages, Compilers, and Tools for Embedded Systems*, volume 1474 of *Lecture Notes in Computer Science*, pages 51–64. Springer Berlin Heidelberg, 1998.

[10] V. B. Lortz and K. G. Shin. Semaphore queue priority assignment for real-time multiprocessor synchronization. *IEEE Transactions on Software Engineering*, 21(10):834–844, Oct. 1995.

[11] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In 23^{rd} *IEEE Real-Time Systems Symposium (RTSS)*, pages 315–326, Dec. 2002.

[12] M. Saksena and Y. Wang. Scalable real-time system design using preemption thresholds. In 21^{st} *IEEE Real-Time Systems Symposium (RTSS)*, pages 25–34, Nov. 2000.

[13] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In 6^{th} *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 328–335, Dec. 1999.

[14] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In 15^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 351–360, Aug. 2009.