A Practical Sub-Optimal Solution for the Dual Priority Scheduling Problem

Tristan Fautrel*, Laurent George*, Joël Goossens†, Damien Masson* and Paul Rodriguez‡
**Université Paris-Est, LIGM UMR CNRS 8049, UPEM, ESIEE Paris, ENPC*, Paris, France, firstname.name@esiee.fr
†*PARTS*, *Université libre de Bruxelles*, *Faculté des Sciences*, Brussels, Belgium, joel.goossens@ulb.ac.be
‡ *HIPPEROS S.A.*, Brussels, Belgium, paul.rodriguez@hipperos.com

***Abstract*—**
We consider uniprocessor platforms, the scheduling of synchronous implicit deadline periodic task sets and the dual priority scheme where each task is assigned two fixed priorities. That is, at run time each task starts executing using its primary priority and is promoted if not completed at an intermediate deadline. We present counter-intuitive examples illustrating how difficult this scheduling problem is. We propose a preprocessing approach to remove from the scheduling problem lowest priority viable tasks as defined by Audsley's procedure. We revisit one solution called $\mathsf{RM}+\mathsf{RM}$ conjectured optimal. We propose a procedure to compute promotion deadlines based on multiple simulations over an hyperperiod called FDMS. That solution has an exponential time complexity but an experimental success ratio of 100%. Then we propose a new sub-optimal solution to assign priorities called $\mathsf{1/RM}+\mathsf{RM}$ along with a very simple promotion deadline assignment scheme called RML for which no simulation are required, the procedure is simple and the success ratio is close to 99.99%. We show that the method is fast and scalable to very large task sets which makes it ideal for practical applications.

***Index Terms*—real-time systems, uniprocessor, scheduling, dual-priority**

I. Introduction

In this research we study the scheduling of synchronous implicit deadline preemptive hard real-time periodic task sets upon uniprocessor systems using the dual priority scheme where each task is assigned *two* fixed priorities (before and after its promotion deadline). At run time, each task instance —*job* in the following— starts executing using its first priority and if not completed when reaching its (intermediary) promotion deadline, continue its execution at a second priority.

In the context of uniprocessor scheduling, two scheduling algorithms have been very much studied: one in the class of fixed task priority (FTP) where Rate Monotonic (RM) is optimal for implicit deadline task sets in this class of schedulers and one in the class of fixed job priority (FJP), where Earliest Deadline First (EDF) is optimal (see [1] for details).

Regarding the utilization bound —the largest utilization value such that all systems with lower or equal utilization are schedulable— RM only achieves 69% in the FTP class while EDF is optimal in the FJP class with an utilization bound of 100%.

This work was supported by "Wallonie-Bruxelles International et du Fonds de la Recherche Scientifique, du Ministère Français des Affaires étrangères et européeens, du Ministère de l'Enseignement supérieur et de la Recherche dans le cadre des partenariats Hubert Curie".

Some research has been done to overcome the sub-optimality of RM w.r.t FJP algorithms in general while retaining other characteristics. It has been shown that when periods are harmonic, the processor utilization bound of RM is 100% as well [2]. Considering the general case, when no constraint is imposed on the periods, the *dual priority* approach was introduced in 1993 [3]. The dual priority approach is interesting as it is *conjectured* that it schedules any system that the EDF scheduler can schedule but with at most one promotion by job while EDF may require a larger number of promotions per job as discussed in [4] and [5] (at most $n-i$ promotions for a task τ_i if tasks are indexed by increasing deadlines and if n is the total number of tasks). Moreover, with dual priority, all the jobs of a same task share the same relative promotion deadline (relative to their release), which is suitable to ease the implementation, the anomaly detection and more generally to overcome commonly reported weakness with practical uses of EDF.

This Research: In this document, we revisit dual priority scheduling for uniprocessor systems with implicit-deadline periodic task sets. Moreover we present counter-intuitive examples illustrating how difficult the scheduling problem is. We first propose a preprocessing approach in Section IV to reduce the number of task to consider. Then, we propose in Section V a solution (of high complexity) to compute promotion deadlines for a conjectured optimal dual priority scheduling algorithm called $\mathsf{RM}+\mathsf{RM}$. Then, we present in Section VI a novel dual priority scheduling named $\mathsf{1/RM}+\mathsf{RM}$. This sub-optimal solution enable us to compute promotion deadlines with a smaller complexity than for $\mathsf{RM}+\mathsf{RM}$. In $\mathsf{1/RM}+\mathsf{RM}$ scheduling, task priorities before promotion are defined by the inverse of RM while the priorities after promotion are defined by RM. Finally, we present some simulation results in Section VII for $\mathsf{1/RM}+\mathsf{RM}$ showing that it has a success ratio of **99.99%**. We conclude and present future works in Section VIII. To the best of our knowledge this the first scalable and near-optimal technique regarding the state of the art. Because of its very high success ratio and its reasonable complexity real applications will benefit from that technique. In particular the implementation of that technique into the RTOS Hipperos is in progress.

II. Definitions and hypothesis

In this work we consider the scheduling of n periodic implicit-deadline tasks denoted as $\tau_1, \ldots, \tau_n$. Each task τ_i is characterized by a Worst Case Execution Time C_i, a period of T_i and a deadline which is equal to the period (implicit-deadline tasks). In addition, each task receives (at design time) an (intermediary) promotion deadline S_i ($0 \leq S_i \leq T_i$) and two priorities P_i^1 and P_i^2, before and after promotion, respectively. Any job of τ_i receives its first priority P_i^1 when released. If the job is not completed after S_i time units, the job is *promoted* and receives its second priority P_i^2. In the particular case where $S_i = T_i$, this corresponds to the classical *single* priority task model that we extend here by assuming $P_i^1 = P_i^2$ (with no promotion for τ_i). The particular case where $S_i = 0$ is similar, although naturally the task starts executing with priority P_i^2, and the value of P_i^1 is irrelevant. The dual priority scheduler requires $2 \times n$ *distinct* priorities (two static priorities per tasks). Because all priorities are distinct, there is no need for a tie-breaking mechanism.

Assumptions: In this work we assume that the tasks periods are distinct, the tasks are indexed according to the rate monotonic order ($T_1 < T_2 < \cdots < T_n$). We consider only the synchronous scenario where all the tasks release their first job at time $t = 0$. We assume that the task set utilization U is defined as follows $U \overset{\text{def}}{=} \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$. Lastly, we consider a dual priority scheme with *two distinct* priority bands (low band, high band) i.e., all the priorities after promotion are higher than all priorities before promotion. Without loss of generality we assume the range of the priorities before promotion is $[n+1, 2n]$ and the range of the priorities after promotion is $[1, n]$. Please notice that we assume that the lowest priority of the system is $2n$ while the highest priority is 1.

III. Related Work

The seminal work [3] of Burns et al. presents the dual priority assignment, which is a *minimal* dynamic priority scheme for synchronous (all first jobs of the tasks are released synchronously), periodic and implicit-deadline task systems. Examples motivate the interest of dual priority scheduler: i.e., systems not schedulable using the optimal FTP assignment (RM) while a feasible solution exists for the dual priority assignment (i.e., with at most one promotion per task). The authors conjectured that for any task set with total utilization not larger than 100% there exists a dual priority assignment (and a promotion deadline assignment) that is schedulable.

Davis and Wellings [6], [7] showed the interest of the dual priority scheme to improve the scheduling of *soft* real-time tasks and *hard* real-time tasks *jointly*. The authors propose an acceptance test for an on line admission of soft/firm real-time tasks. In the same vein, Banùs et al. [8] present an adaptation of the dual priority scheduling algorithm to schedule both hard real-time periodic tasks and soft-aperiodic tasks in shared memory multiprocessor systems. Unfortunately, those works cannot be used in our framework since we consider the scheduling of a single set of *hard* real-time tasks.

More recently, Burns [9] proved the optimality of the dual priority (the conjecture) for systems composed of *two* tasks. This remains an open question for the general case, as no counter-example has been yet found.

In a previous work [4], we reported (mainly negative) results regarding dual priority scheduling (see next section) that confirm the singularity of the dual priority scheduling problem.

A. Previous Results Presented in [4]

We report here (counter-intuitive) properties which illustrate that dual priority scheduling is a scheduling class which differs from FTP and FJP.

Property 1 (Response time of the first job [4]). *Under* synchronous *implicit-deadline task set, using dual priority, the response time of the first job is* not *necessarily the largest one.*

Property 2 (The first busy period [4]). *Under* synchronous *implicit-deadline task set, the first busy period is* not *a feasibility interval for the dual priority scheduling.*

Property 3 (No critical instant [4]). *Under dual priority scheduling, the synchronous case is* not *always the worst case.*

Property 3 justifies our assumption that the task set is synchronously released. Indeed, there is no known feasibility condition or exact test for the dual priority scheduling but if we limit our investigation to the synchronous case, it is still possible to simulate the system execution over a hyperperiod (i.e., from time origin to the least common multiple of the task periods), which is most likely not true considering possible task offsets.

Lastly, we introduced in [4] a new class of scheduling FP^k, a fixed priority scheduling that requires at most k promotions at k fixed times w.r.t the release time of a job of a task. We show that dual priority and EDF scheduling are particular cases of FP^k. Finally, we analyze EDF scheduling trying to study how far it is from a dual priority scheduler in terms of promotions. We show that EDF may lead to more than one promotion per job. Hence, trying to mimic EDF is probably not the best way to find priorities and promotion deadlines for the dual priority scheduling.

B. Other Conjectures

Conjecture 4 (Maximal Utilization Bound [9]). *For any task set with more than two tasks with total utilization less than or equal to* 100%*, there exists a dual priority assignment (and a promotion deadline assignment) that will meet all deadlines.*

In addition to Conjecture 4, a conjecture on the priority assignment scheme for dual priority scheduling with RM+RM is as follows:

Definition 5 (RM + RM dual priority scheduling). *The priorities before promotion ($P_1^1, \ldots, P_n^1$) are given by* RM*, the priorities after promotion ($P_1^2, \ldots, P_n^2$) are given by* RM *such that all the priorities after promotion are higher than all priorities before promotion.*

Conjecture 6 (Optimality of $\mathsf{RM}+\mathsf{RM}$)**.** *$\mathsf{RM}+\mathsf{RM}$ priority ordering is optimal for the dual priority problem.*

IV. Preprocessing

In this section we show that, *without any loss of generality*, we can remove, from the scheduling problem, tasks that are Lowest Priority Viable (LPV, as defined by Audsley, see [10] and Definition 10). Our main property, Theorem 11, builds upon the work of Audsley but requires first to prove that dual priority schedules are C-sustainable. To the best of our knowledge, while the literature addresses this question for FTP, FJP and LLF even upon multiprocessors, the property was not formally proved for dual priority upon *uni*-processor.

Definition 7 (C-sustainable)**.** *A scheduling algorithm is said to be* C-sustainable *if decreasing a execution time of any job will not increase the response time of any job.*

Following the argument of Han & Park [11] we consider two job sets $G = \{g_1, g_2, \dots, g_n\}$ where $g_\ell = (a_\ell, e_\ell, d_\ell, s_\ell, p^1_\ell, p^2_\ell)$ the characteristics correspond to the arrival time, computation time, deadline, promotion times, the priority before and after promotion, respectively, and $G' = \{g'_1, g'_2, \dots, g'_n\}$ where $g'_\ell = g_\ell$ for $\ell \neq i$ and $g'_i = (a_i, \mathbf{e_i - 1}, d_i, s_i, p^1_i, p^2_i)$. I.e., g'_i is identical to g_i except the execution time requirement is smaller by 1 time unit. In the following, $f(g_\ell)$ will denote the completion time of the job g_ℓ in the schedule of the set G.

C-sustainability is a consequence of the following property.

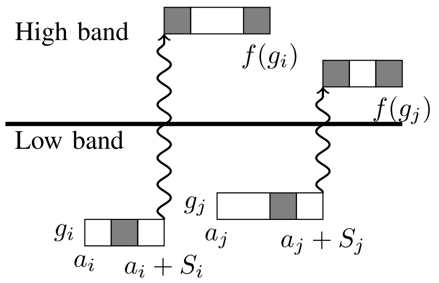


Fig. 1: Schedule of the set G.

Lemma 8. *$f(g'_\ell) \le f(g_\ell)$, for $1 \le \ell \le n$.*

Proof. First consider the schedule of the set of jobs G using the dual priority rule (see Figure 1). Now let consider the schedule of G': the job g'_i will complete sooner and will leave the time unit $f(g_i)$ available for the scheduling of other jobs. First notice that if there is no active job at time $f(g_i)$ the property is trivially true. Otherwise we have to consider the highest priority and active job at time $f(g_i)$, i.e., the job g'_j (or g_j) in our example. The job g'_j will execute in the same "*envelope*" using the same instants than the schedule of G except there is an extra available unit at time $f(g_i)$. Job g'_j will consequently finish its execution *sooner* ($f(g'_j) < f(g_j)$), leaving the time unit $f(g_j)$ available for other active jobs (if any). Repeating the argument we prove the property. $\square$

Corollary 9. *The dual priority scheduler is C-sustainable.*

Proof. This is a direct consequence of Lemma 8 $\square$

Definition 10 (Lowest Priority Viable (LPV))**.** *A task τ_i is said to be LPV iff all its jobs meet their deadline when:*

- *task τ_i has the lowest priority*
- *the other tasks have any higher priority*
- *when scheduling tasks $\tau_j \neq \tau_i$, the scheduler considers deadlines as soft, i.e. continues to schedule until completion*

Theorem 11. *Suppose that τ_i is LPV. Then, there exists a feasible dual priority assignment (P^1_i, P^2_i, S_i) for the task set $\tau \overset{\mathsf{def}}{=} \{\tau_1, \dots, \tau_\ell\}$ if and only if there exists a feasible dual priority assignment for of $[\tau \setminus \{\tau_i\}]$.*

Proof. Suppose that the following dual priority assignment p is feasible for τ:

$$p = \big\langle (P^1_1, P^2_1, S_1), (P^1_2, P^2_2, S_2), \dots, (P^1_\ell, P^2_\ell, S_\ell) \big\rangle$$

then a second priority assignment is feasible as well:

$$\begin{aligned} p' =< &(P^1_1, P^2_1, S_1), \dots, (P^1_{i-1}, P^2_{i-1}, S_{i-1}), \\ &(P^1_i = -\infty, P^2_i = \text{undefined}, S_i = T_i), \\ &(P^1_{i+1}, P^2_{i+1}, S_{i+1}), \dots, (P^1_n, P^2_n, S_n) > \end{aligned}$$

Since τ_i is LPV, this task is schedulable without promotion and with the lowest priority. More formally by choosing $(P^1_i = \infty, P^2_i = \text{undefined}, S_i = T_i)$. (Following this Theorem we will see exactly how to assign a unique priority to each LPV task). The other dual priority tasks $\tau_1, \dots, \tau_{i-1}, \tau_{i+1}, \dots, \tau_\ell$ remain schedulable with the very same priority (and promotion deadline) assignment since dual priority schedulers are C-sustainable (Corollary 9). $\square$

Based on Theorem 11 we designed Algorithm 1 to obtain from a task set of N tasks the subset σ of n dual priority tasks after the preprocessing and the subset τ' of $N - n$ LPV tasks and their relative priorities.

Please notice that if initially the task set is RM-schedulable (e.g. if $U \le 69\%$) the preprocessing removes *all* tasks, the system is obviously dual priority schedulable. In the following we assume that the remaining task set (after the preprocessing phase) is not empty.

The preprocessing is interesting regarding the number of promotions since the LPV tasks (if any) are scheduled in *background* without any promotion.

Using this approach, a task set of N periodic tasks can be decomposed in two sets: one corresponding to $n \le N$ tasks that are dual priority tasks, and one corresponding to $N - n$ tasks that are LPV. For the dual priority tasks, their priority before and after promotion will be chosen in $[n+1, 2n]$ (and in $[1, n]$, respectively) according to the dual priority scheduling chosen ($\mathsf{RM} + \mathsf{RM}$ algorithm in Section V or $\mathsf{1/RM} + \mathsf{RM}$ in Section VI). By doing so, all tasks after promotion (if promoted) have a higher priority than non promoted jobs.

Algorithm 1 Preprocessing

```
Require: $\sigma$, a set of $N$ tasks
  Boolean continue $\leftarrow$ TRUE
  $\tau' \leftarrow \emptyset$ {lowest priority tasks}
  $j \leftarrow 0$
  repeat
    continue $\leftarrow$ FALSE
    for all $\tau_i \in \sigma$ do
      if $\tau_i$ is LPV then
        $\sigma \leftarrow \sigma \backslash \{\tau_i\}$
        $j \leftarrow j + 1$
        $\tau'_j = \tau_i$ {add an element to $\tau'$}
        continue $\leftarrow$ TRUE
      end if
    end for
  until !continue
  $n \leftarrow |\sigma|$ {the number of dual priority tasks}
  for all $i$ s.t. $1 \leq i \leq j$ do
    $P'_i = 2n + j - i + 1$ {the priority of $\tau'_i$}
  end for
  return $\sigma$, a set of $n$ dual priority tasks with $n \leq N$
  return $\tau'$, a set of $j = N - n$ lowest priority tasks with
  the priority $P'_1, \ldots, P'_j$, respectively
```

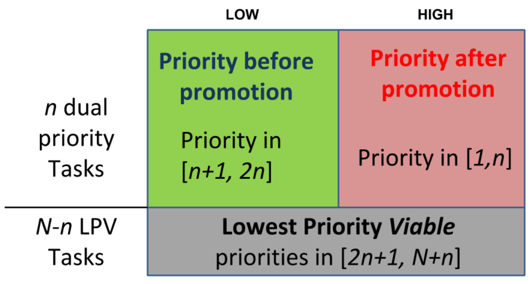


Fig. 2: The case of n dual priority tasks among N.

Concerning the $N-n$ LPV tasks, their priority after promotion is not used as tasks are not promoted and their priority (before promotion) is chosen in $[2n+1, N+n]$. Figure 2 describes the principle of the priority assignment we propose.

V. RM+RM & THE PROMOTION ASSIGNMENT FDMS

This section aims to present our investigation on the special priority assignment case called RM + RM (see Definition 5). The preprocessing detailed in Section IV can or cannot be applied. Its main interest under this priority assignment scheme is to potentially decrease the number of tasks to consider, and then lower the algorithmic complexity of the computations.

With RM + RM, tasks priorities before promotions are assigned following the RM rule in a low-priority-band while the same is done in a high-priority-band to assign priorities after promotion. As presented in Section III, Conjecture 6 supposes that RM+RM is an optimal dual priority assignment under our assumptions.

In addition to proving this conjecture, a remaining challenge consists in designing an algorithm that find the promotion deadlines. We present in this section a naive brute force algorithm called First Deadline Missed Strategy (FDMS) to address this challenge.

We also conjecture that this algorithm is optimal (in the sense that it always finds suitable promotion deadlines if the Conjectures 4 and 6 are true). Unfortunately, it requires a feasibility test and the only one available for now is a complete simulation of the task set under study over an hyperperiod. Moreover such a simulation is only viable for the synchronous scenario where all the jobs of the tasks are first released at time 0.

A. First Deadline Missed Strategy (FDMS)

Algorithm 2 simulateDual()

```
Require: $\sigma$, a priority-ordered set of $n$ tasks
Require: $\mathcal{P}$, a dual priority assignment rule
Require: $S$, a set of promotion deadlines
  Simulates until a deadline miss occurs or until the end of
  the hyperperiod
  if Simulation ends because $\tau_i$ misses a deadline then
    return (FALSE, $i$)
  else
    return (TRUE,-1);
  end if
```

Algorithm 3 FDMS to set promotion deadlines

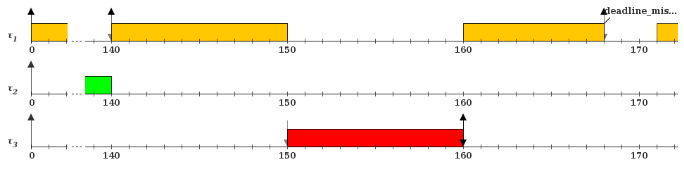
```
Require: $\sigma$, a priority-ordered set of $n$ tasks
Require: $\mathcal{P}$, a dual priority assignment rule
  Set S $\leftarrow (T_1, T_2, \ldots, T_n)$
  (Boolean success, Integer $i$) $\leftarrow$ simulateDual($\sigma, \mathcal{P}, S$)
  while !success do
    if $S_i$ equals 0 then
      return FAILURE
    end if
    $S_i \leftarrow S_i - 1$
    (Boolean success, Integer $i$) $\leftarrow$ simulateDual($\sigma, \mathcal{P}, S$)
  end while
  return $S$
```

This practical approach formalized in Algorithm 3 (and in Algorithm 2) consists in assigning $S_i = T_i$ for all tasks τ_i, and then addressing the deadline misses in a chronological order. In order to do so, the promotion deadline for the first faulty task is decreased by one. If it was already equal to zero, the algorithm fails and the system is declared not schedulable, but we conjecture that this cannot happen.

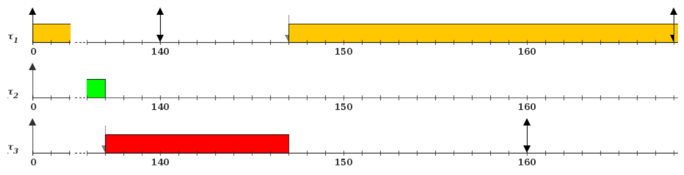
However, the theoretical complexity of this approach can be significantly high, as a deadline miss can happen for every job deadlines in $[0, H)$ (H denotes the hyperperiod, i.e. least common multiple of the task periods), requiring to update the value of each task τ_i promotion deadline at most T_i times (from Algorithm 3). In the case of implicit deadlines, this

	C_i	T_i
τ_1	21	28
τ_2	15	100
τ_3	16	160

(a) System Data



(b) $(S_1, S_2, S_3) = (28, 100, 150)$



(c) $(S_1, S_2, S_3) = (7, 100, 137)$

Fig. 3: FDMS example

lead to a complexity equal to $(\sum_{\forall i} \lceil \frac{H}{T_i} \rceil \times T_i) \times H \leq n(H)^2$ (where $H \overset{\text{def}}{=} \text{lcm}_{i=1}^{n} T_i$). In the case of prime periods higher than 2, $H \geq 2^n$, the complexity of FDMS is then exponential.

This algorithm can be optimized for a job J_i of τ_i missing its deadline, when setting S_i to $S_i - 1$ will not fix the deadline miss of J_i (i.e., the lateness is greater than 1). For example, when the remaining execution time of J_i is greater than 1 (leading to S_i decreasing by at least this remaining execution time), or when another higher priority task is promoted before S_i (leading to S_i decreasing by at least the execution time of the promoted job after its promotion plus one). However, efficient implementation of this algorithm is not the subject of this paper.

B. Example

We describe in this section the steps to compute suitable promotion deadlines for the priority scheme RM + RM with FDMS on the example given by Figure 3. The first step is to set (S_1, S_2, S_3) to $(28, 100, 160)$ ($\forall i, S_i = T_i$) and to simulate the system as long as no deadline is missed.

This first deadline miss occurs at time 160, and concerns the first job of τ_3. It has been executed only during 6 time units (between time 78 an 84). Hence 10 execution times are missing for its execution. This leads setting (S_1, S_2, S_3) to $(28, 100, 150)$ (see Figure 3b —strictly following Algorithm 3 requires setting $S_3 = 159$, simulate, notice that the same deadline is missed, set $S_3 = 158$ and so on until $S_3 = 150$).

The next simulation shows that a deadline is missed by τ_1 at time 168. To make this job complete before its deadline, a promotion deadline must be set at time 157, which corresponds to set $S_1 = 17$. However, strictly following Algorithm 3 will decrease S_3 one by one from 28 to 19. And that point that the τ_3 deadline at time 160 is missed again. Let us so set for now (S_1, S_2, S_3) to $(19, 100, 150)$. Reducing S_3 to 149 is enough to solve the deadline missed at time 160, but the deadline of τ_1 at time 168 is still missed. So the next steps of the algorithm will lead to successively decrease by one S_3 then S_1 until we obtain $(S_1, S_2, S_3) = (7, 100, 137)$ (see Figure 3c).

Finally, the first deadline miss now occurs at time 500, and the faulty task is τ_2. The next steps will decrease S_2 one by one with no effect until $S_2 = 83$ because τ_1 is executing in the high priority band between time 483 and time 500. However, setting S_2 to 82 will permit τ_2 to met this deadline. We now have $(S_1, S_2, S_3) = (7, 82, 137)$.

The next deadline miss concerns a job of τ_3 at time 640. It can be resolved by setting $S_3 = 136$. Again, τ_3 misses another deadline at time 1760, leading the algorithm to set $S_3 = 132$, then another one at time 2240 leading to set $S_3 = 131$ and a finale one at time 3360 solved by setting $S_3 = 130$.

With $(S_1, S_2, S_3) = (7, 82, 130)$, the system is feasible with RM + RM dual priority scheduling.

C. Insights Regarding the Optimality

We strongly conjecture that FDMS is an optimal strategy. The following is not a formal proof, but gives our insights on this property.

A first observation is the fact that in order to resolve a particular deadline miss (say the job J_i) only two actions (on the promotion deadlines) can resolve that deadline miss:

1) reduce the value for S_i,
2) increase the value for an other task.

Since FDMS algorithm initially assigns S_i to T_i and decreases it only if needed to solve a particular deadline miss, the second action, when possible, will obviously lead to miss again that very same deadline.

Then decreasing the value of a particular S_i cannot lead to miss the deadline of a previous job of τ_i or of a task in $\tau_{i+1}, \ldots, \tau_n$ which were schedulable without any promotions ($S_k = T_k$). Indeed, if previous lower priority jobs were feasible without promotions, the promotion of J_i and subsequently previous jobs of τ_i will not add interference because they are executed and schedulable in the low priority band. If previous jobs of τ_i were feasible without promotions, lower S_i can only make them complete earlier.

However, it can:

- (case 1) lead to make an higher priority task[1] miss its deadline (as it is the case in the example described in the previous sub-section, see Sub-Figure 3c),
- (case 2) lead to make a job of a lower priority task[1] miss its deadline in the case where this job only reached its deadline because of a promotion given earlier to the task by the algorithm (again it is the case in the example given, for the job of τ_3 with a deadline at time 160).

Case 1: the algorithm will be able to fix this kind of induced deadline miss by lowering the promotion deadline of this higher task[1] (let us denote it τ_k with a promotion value of

[1]according to RM priority ordering

S_k). This will possibly lead to encounter the case 2. In the worst case, the algorithm will successively lower values of S_i and S_k (as it is the case in the developed example) until both task execute entirely in the high band ($S_i = S_k = 0$), with relative priorities RM, for which τ_k were feasible in the first place.

Case 2: again, this will lead to loop with successively lower values of promotion deadlines for two tasks, which will be ultimately resolved in the worst case by executing both tasks in the high band.

Note also that both cases can apply to several task, but since we address the first deadline miss, this does not change the given arguments.

We can conclude that if it is not guaranteed that the initial deadline miss is solved, the algorithm will not introduce other deadline misses in the past (relative to the addressed one) which it cannot solve. Taking deadline misses in a chronological order should then ensure us to progress until the end of the hyperperiod, making the whole system feasible if possible. Moreover, if a promotion deadline for a task reaches 0 and if a job of this task misses its deadline, all other greater values for this promotion deadline have been tested before (since the algorithm decreases the promotion deadline one unit at a time).

In our opinion, the last missing argument to prove the optimality is the demonstration that taking deadline misses by chronological order rather than, e.g., task by task, leads to minimizing the introduced constraints.

VI. 1/RM+RM & THE PROMOTION ASSIGNMENT RML

We present in this section our practical sub-optimal solution for the dual priority scheduling problem. Sub-Section VI-A presents the priority assignment $1/\text{RM} + \text{RM}$, Sub-Section VI-B introduces the promotion deadline assignment rule RML and Section VI-C presents counter examples that prove that none are optimal.

A. *1/RM + RM*

Definition 12 ($1/\text{RM} + \text{RM}$ dual priority scheduling)**.** *The priorities before promotion ($P^1_1, \dots, P^1_n$) are given by the opposite of RM, i.e. the longer the period, the higher the priority. The priorities after promotion ($P^2_1, \dots, P^2_n$) are given by RM such that all the priorities after promotion are higher than all priorities before promotion.*

The priority values of n dual priority tasks with $1/\text{RM}+\text{RM}$ are thus split in two distinct priority bands according to the principles described in Figure 2. In the first low priority band (before promotion), tasks are scheduled inverse to RM, i.e., tasks with higher periods have higher priorities with priority chosen in $[n+1, 2n]$. This priority ordering is denoted $1/\text{RM}$. After promotion, tasks are assigned priority in a second high priority band according to RM priority ordering with priorities chosen in $[1, n]$. The *LPV band* is used for the $N-n$ tasks that are LPV. Those tasks are not promoted and their intra-band ordering is determined by the preprocessing with priorities chosen in $[2n+1, N+n]$.

	P^1_i	P^2_i
τ_1	6	1
τ_2	5	2
τ_3	4	3

TABLE I: Example of $1/\text{RM} + \text{RM}$ priority assignment with $T_1 \leq T_2 \leq T_3$ and no LPV task. Note that all pre-promotion priorities are weaker than post-promotion priorities, and that the promotion of τ_3 is irrelevant.

For instance if $T_1 \leq T_2 \leq T_3$ and none of the tasks are LPV ($n = N = 3$), then $1/\text{RM}+\text{RM}$ defines the dual priority assignment of Table I.

More generally, as the tasks are indexed according to rate monotonic priority ($T_1 \leq T_2 \leq \cdots \leq T_n$), then we have $P^2_i = i$ and $P^1_i = 2n - i + 1$.

The main interest of this priority assignment strategy is that once the promotion for τ_1 is set to make it schedulable, no promotion change on other task can change its schedulability. More generally, once the subset of tasks $\tau_1 ... \tau_k$ is found schedulable, no promotion assignment on tasks $\tau_{k+1}, \dots, \tau_n$ will jeopardize that. This is due to the fact that tasks $\tau_{k+1}, \dots, \tau_n$ have higher priorities in the low band and lower in the high band. Hence, a decrease in promotion priority for one of this task will have no effect on the interference on any particular task in $\tau_1, \dots, \tau_k$: if it is not promoted, it still has a relative lower priority, if it is promoted, it still has a relative higher priority.

B. *The RM Laxity promotion deadline assignment (RML)*

Based on the main interest of this priority assignment, a promotion deadline assignment algorithm for $1/\text{RM} + \text{RM}$ should assign promotions deadline task by task following the order of their RM priorities. An optimal algorithm can then consist in a variant of the FDMS, where we first set τ_1 promotion deadline as big as possible to keep it schedulable, then proceed the same way for τ_2 and so on. However, this will require a lot of simulations to determine the schedulability.

A much simpler and still quite effective way to proceed consist in assigning each task a promotion deadline equal to its laxity when scheduled with RM.

Definition 13 (RM Laxity)**.** *The RM Laxity of a task is given*

- *by the distance between its completion and its deadline when the system is scheduled with RM if the task is RM-schedulable,*
- *0 is the task is not RM-schedulable.*

To compute the RM laxity experienced by each task, we can use the fixed-point iterative approach first introduced in [12] to find the worst case response time of each task scheduled with RM. However, one has to note that we can stop the fixed-point computation as soon as it converges or if the task deadline is reached.

$$S_i = \max(0, T_i - R_i) \qquad (1)$$

with

$$R_i = \min_{t>0} \left\{ t = \sum_{k<i} \left\lceil \frac{t}{T_k} \right\rceil C_k + C_i \right\} \quad (2)$$

To the best of our knowledge, this is the first dual priority promotion deadline assignment scheme not based on the simulation of a task system on a hyperperiod.

In comparison, the execution time required to find appropriate promotion deadline according to our scheme is bounded by $n \times \max_{i=1,\ldots,n} T_i$ as we do not need to compute the exact worst case response time R_i for implicit deadline periodic tasks as in [12] as S_i is the maximum value between 0 and $T_i - R_i$. The computation of R_i is done by a fixed point equation that can be stopped as soon as $T_i - R_i \leq 0$. Compared to the complexity of the worst case response time computation (NP-hard in the worst case [13]), it is pseudo polynomial in our case.

We show in the next sub-section (VI-C) that $1/\mathsf{RM} + \mathsf{RM}$ priority assignment for promoted tasks is not optimal, i.e. there are systems that are (dual priority)-feasible but not schedulable with this technique (Figure 6). The RML promotion deadline scheme is also non optimal, as it exists taskset schedulable with $1/\mathsf{RM} + \mathsf{RM}$ but not schedulable with promotions given by RML (Figure 5). Finally, we also show that the preprocessing is important for this priority assignment by showing an example where a task set is not schedulable when no preprocessing is done and becomes schedulable with the preprocessing (Figure 4).

However the computational complexity is very interesting regarding other existing approach ones and we give an experimental assessment of this technique in Section VII.

C. Counter-examples for $1/\mathsf{RM} + \mathsf{RM}$

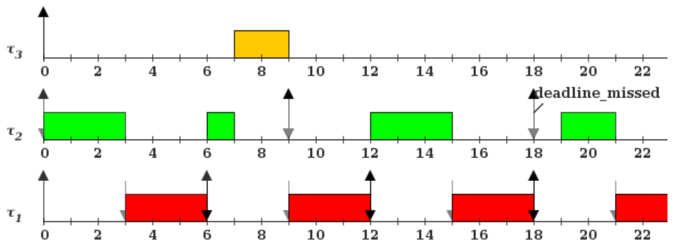
Figure 4 depicts an example showing that the last tasks should be scheduled in background when they are feasible with RM. Promotion deadlines are assigned according to the RML (values are given in Table 4a along with other example data: $S_1 = 3$, $S_2 = S_3 = 0$). As we can see in Sub-figure 4b, if we apply the $1/\mathsf{RM} + \mathsf{RM}$ scheme considering all the tasks, then τ_3 will execute at the highest low-band-priority between time instants 7 and 9, and then it follows that τ_2 will miss its deadline at 18. Note that no promotion deadline assignment can solve this issue. However, applying the preprocessing will identify task τ_3 as the only LPV task and so lead to schedule it in background. Applying RML to the subset (τ_1, τ_2) gives suitable promotion deadlines $(S_1 = 3, S_2 = 9)$ for $1/\mathsf{RM} + \mathsf{RM}$. The system is then schedulable, as illustrated by Sub-figure 4c.

Figure 5 depicts an example showing that RML is not optimal for the priority assignment $1/\mathsf{RM} + \mathsf{RM}$. Indeed, as shown in Sub-Figure 5b, the system is not schedulable with RML, however with the set of promotion deadlines $S_1 = 38, S_2 = 77, S_3 = 183$, the system is $1/\mathsf{RM} + \mathsf{RM}$ schedulable.

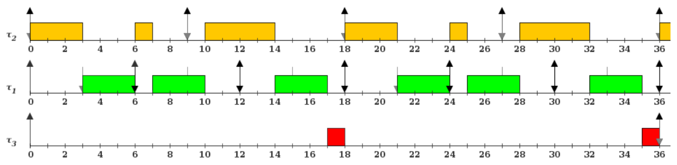
Figure 6 depicts an example showing that $1/\mathsf{RM} + \mathsf{RM}$ is not optimal. First, notice that the system is not feasible

	C_i	T_i	RM laxity
τ_1	3	6	3
τ_2	4	9	0
τ_3	2	36	0

(a) System Data



(b) τ_3 part of the problem

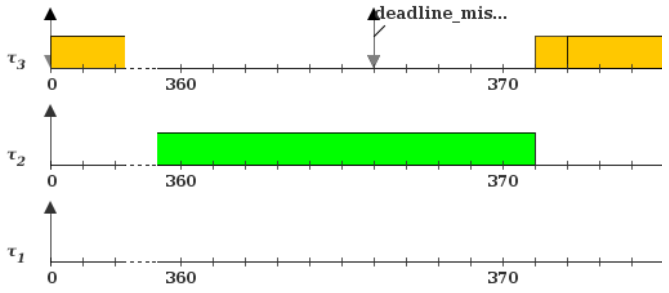


(c) τ_3 scheduled in background

Fig. 4: Example showing why RM feasible low priority tasks should be run in background

	C_i	T_i	RM laxity
τ_1	13	51	38
τ_2	83	128	6
τ_3	16	183	0

(a) System Data

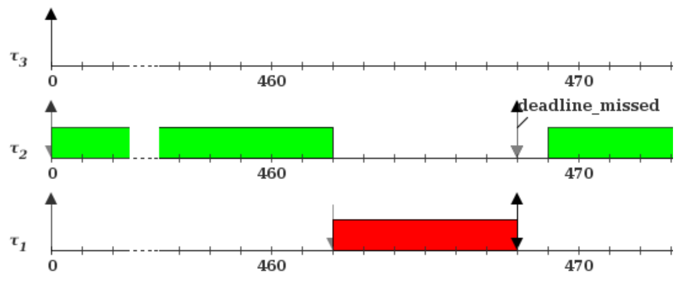


(b) $S_i = \mathsf{RM}$ laxity$_i$: τ_3 is not schedulable, however, the system is schedulable with $S_1 = 38, S_2 = 77, S_3 = 183$

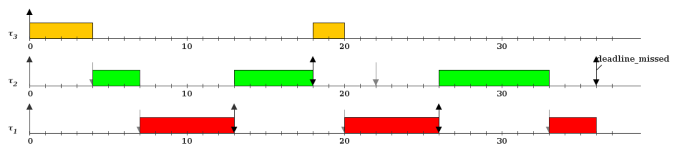
Fig. 5: Example showing that $S_i = \mathsf{RM}$ laxity$_i$ is not optimal while the priorities are $1/\mathsf{RM} + \mathsf{RM}$

	C_i	T_i	RM laxity
τ_1	6	13	7
τ_2	8	18	0
τ_3	6	86	0

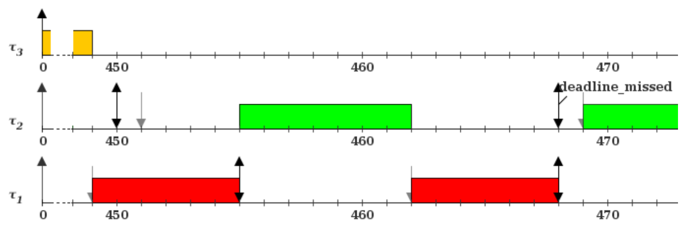
(a) System Data



(b) With $S_i =$ RM laxity there is a deadline miss at 468



(c) $S_2 = 4$ makes the first job of τ_2 schedulable, but not the second one



(d) $S_2 = 1$ there is a deadline miss which is unavoidable, even with $S_2 = 0$

Fig. 6: Example showing that $1/\text{RM} + \text{RM}$ is not optimal

with the promotion deadlines given by RML (Sub-Figure 6b). Once S_1 assigned to 7, to make τ_2 schedulable, we need to permit its first instance to complete before its deadline. To do so, S_2 must be not greater than 4 (Sub-Figure 6c). Then to schedule its second job the constraint is $S_2 \leq 1$. But then, a deadline is missed in 468 (Sub-Figure 6d). There is no solution because even a promotion deadline in 0 does not permit to gain any time slot for this instance (τ_1 is executing and is already promoted). One could argue that we could lower the value of S_1. That is true, but no value makes the system schedulable. This is due to the fact that τ_2 is not feasible with RM. The need for τ_1 to steal time from τ_3 forces us in a situation where τ_1 and τ_2 relative priorities are RM. This task set however is schedulable with $\text{RM} + \text{RM}$ using the promotion deadlines $(S_1, S_2, S_3) = (13, 17, 84)$.

VII. Experiments

The $1/\text{RM}+\text{RM}$ technique with promotion deadlines equal to RM laxity (RML) has been tested extensively. We conducted two series of experiments which differ mainly by the

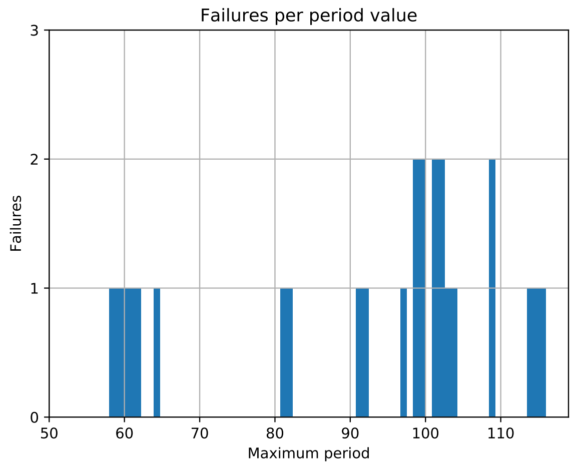


Fig. 7: Failures of the RML technique with respect to the longest period of the taskset. More than 1 800 tasksets were tested for each maximum period value in the interval 50 to 120. Tasksets had 3 to 8 tasks and the smallest period was fixed at 40. The tasksets were generated with total utilization random chosen in the interval 0.9 to 1.

taskset generation procedures. The frequency of unschedulable tasksets found in both cases suggests that the success ratio of the scheme ($1/\text{RM} + \text{RM,RML}$) is very close to 1. All the results were gathered using a simulation tool which has been made freely available for use and modification (see [14]).

For the first set of experiments, we generated tasksets of 3 to 8 tasks where the smallest task period was always 40 and the longest task period varied from 50 to 120. Total taskset utilization varied between about 0.9 and 1. We generated a total of 777 000 such tasksets and recorded whether the ($1/\text{RM}+\text{RM,RML}$) successfully schedule them to their hyperperiod. Results are shown in Figure 7. The absolute number of unschedulable taskset per period value is between 0 and 2 for each period value in the interval. The total number is 27 which is equivalent to have a success ratio of about 99.995%. The spread of unschedulable tasksets suggests that they are more likely to be generated when the ratio of periods of a taskset is greater than two. However, there is not enough data to show that this is a statistically significant pattern. More importantly, we have found multiple unschedulable tasksets with all periods in the interval 40 to 64, showing that a ratio between the shortest and the longest period less than two is not a sufficient schedulability condition for ($1/\text{RM} + \text{RM,RML}$).

Note that with these task parameters, the preprocessing technique (described in Section IV) alone is capable of scheduling 49.7% of the generated tasksets. That is, about half of the generated tasksets cannot be scheduled using FTP algorithms. Among those systems that are not schedulable using the preprocessing alone, still about 99.99% are schedulable using ($1/\text{RM} + \text{RM,RML}$). That leaves 27 unschedulable systems. Since RML is not an optimal promotion deadlines assignment

	C_i	T_i	S_i	P_i^1	P_i^2
τ_1	9	40	31	6	1
τ_2	35	54	1	5	2
τ_3	9	74	N/A	4	N/A
τ_1	1	40	39	8	1
τ_2	16	48	31	7	2
τ_3	37	73	2	6	3
τ_4	12	101	N/A	5	N/A
τ_1	1	40	39	10	1
τ_2	7	60	52	9	2
τ_3	27	75	40	8	3
τ_4	35	100	0	7	4
τ_5	17	119	N/A	6	N/A
τ_1	16	40	24	12	1
τ_2	8	40	16	11	2
τ_3	1	60	35	10	3
τ_4	1	66	40	9	4
τ_5	15	76	10	8	5
τ_6	16	101	N/A	7	N/A

Fig. 8: Table of some of the found counter-examples for the RML promotion deadline scheme.

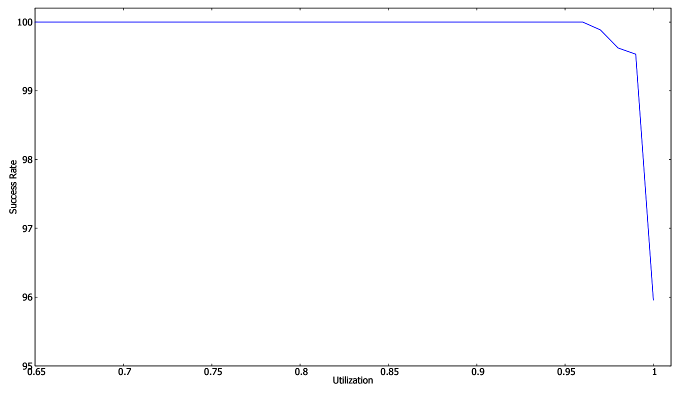


Fig. 9: Success ratio of 1/RM + RM vs taskset utilization after preprocessing

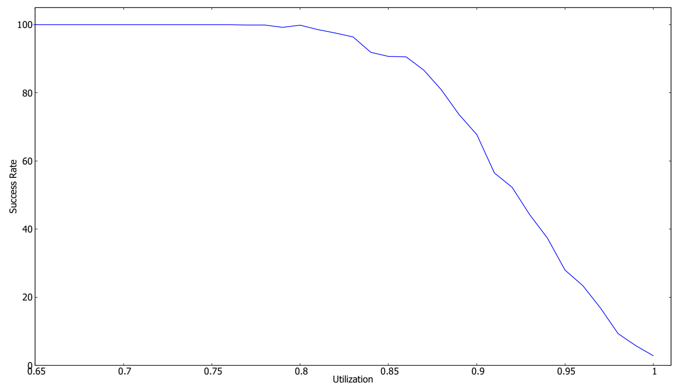


Fig. 10: Success ratio of the preprocessing approach vs taskset utilization.

scheme for $1/\mathsf{RM} + \mathsf{RM}$, we were interested in quantifying its success ratio. We exhaustively searched feasible promotion deadlines for $1/\mathsf{RM} + \mathsf{RM}$ for 10 of these 27 unschedulable systems (the process is heavily time consuming). We found 5 $1/\mathsf{RM} + \mathsf{RM}$-feasible systems and 5 infeasible ones. As these counter-examples to the RML promotion deadline scheme are time-consuming to find, we show some of them in Figure 8. These could help the interested reader to pursue this work.

It is important to note that all the generated systems are schedulable with ($\mathsf{RM} + \mathsf{RM}$,FDMS).

For the second set of experiments, we considered the success ratio of $1/\mathsf{RM} + \mathsf{RM}$ scheduling and the success ratio of the preprocessing approach detailed in Section IV to prune tasks that are LPV for arbitrary task periods. We generated $77\,000$ new tasksets with a UUniFast taskset generator [15] generating arbitrary periods in $[100, 100\,000]$ and processor utilization in $[\ln(2), 1)$. We use two versions of UUnifast, one using an hyperperiod limitation technique described in [16], the other using the classic algorithm. Indeed, the limitation technique tends to generate tasksets with harmonic periods, for which RM is already optimal. However, when the hyperperiod is too long, the simulations cannot be completed until the end, and we so stop them to the minimum between the hyperperiod and 999999. This could introduce false-positive schedulable tasksets, but since the results are consistent between experiments, we do not think that this skew the average results.

Figure 9 shows the success ratio of ($1/\mathsf{RM} + \mathsf{RM}$,RML) as a function of the processor utilization after applying the preprocessing described in Section IV. We therefore compute this success ratio only for tasksets that are not schedulable by RM scheduling. We show that the success ratio is equal to 100% when processor utilization is less than 0.97. The success ratio then decreases when the utilization increases but is still equal to 99.6% when utilization is equal to 0.98. This confirms that ($1/\mathsf{RM} + \mathsf{RM}$,RML) is a very good sub-optimal solution for the dual priority scheduling problem.

Figure 10 shows the performance of the preprocessing approach described in Section IV. For tasksets having a utilization less than 0.8, the preprocessing is able to remove 100% the tasks. The success ratio of the preprocessing is equal to 50% when processor utilization is equal to 0.9. This confirms the interest of applying our preprocessing approach when considering the scheduling of dual priority tasks.

We can conclude two things from these experiments.

First, it is more difficult to find counter examples for the dual priority scheduling problem for randomly uniform generated task set. This is an axis of future work to determine how problematic task sets can be generated more efficiently.

Second, based on our taskset sample, it seems that a very good strategy to schedule a taskset is to first apply our preprocessing, if it remains tasks to schedule with dual priorities, then try ($1/\mathsf{RM}+\mathsf{RM}$,$\mathsf{RML}$). Eventually, in the rare cases where the system is not schedulable with that scheme, then proceed with ($\mathsf{RM}+\mathsf{RM}$,FDMS).

VIII. Conclusion

In this paper, we revisit the dual priority scheduling problem in the case of synchronous implicit deadline periodic tasks. We show with examples that the dual priority problem contradicts well known approaches for fixed priority scheduling to establish feasibility conditions. Then we propose a preprocessing approach to remove from the list of tasks to consider those that are lowest priority viable as defined by Audsley's procedure. Then we revisit one solution called $\mathsf{RM}+\mathsf{RM}$ conjectured optimal in the state of the art. We propose a solution to compute promotion deadlines based on multiple simulations of $\mathsf{RM}+\mathsf{RM}$ scheduling over a hyperperiod. This solution has an exponential time complexity but a success ratio equal to 100% in all the experiments we conducted. Then, we propose a new sub-optimal solution called $1/\mathsf{RM}+\mathsf{RM}$ for which no simulation is needed to compute promotion deadlines. Promotion deadlines are simply set according to a simplified worst case response time computation approach of tasks with RM scheduling (it is simplified because we only need the exact worst case response time when it is lesser or equal to the deadline). We show the benefit of the preprocessing to find feasible solutions for $1/\mathsf{RM}+\mathsf{RM}$. Finally we show that the success ratio of $1/\mathsf{RM}+\mathsf{RM}$ is close to 99.99% with a pseudo polynomial complexity.

As a further work, we would like to optimize the computation of promotion deadlines for $\mathsf{RM}+\mathsf{RM}$ dual priority scheduling with our FDMS approach. We also want to investigate on effective methods to generate task sets unschedulable without dual priority, and that challenge simplest ways to select priorities and promotion deadlines. Then, we plan to formally demonstrate that FDMS strategy is optimal for promotion deadlines computation i.e. if FDMS fails to find promotion deadlines then no promotion deadlines exist. This should help to tackle the problems to show that $\mathsf{RM}+\mathsf{RM}$ is an optimal dual priority scheduling and that dual priority scheduling is optimal for scheduling implicit deadline periodic synchronous tasks. We also aim to find feasibility conditions (sufficient and/or necessary) for the dual priority scheduling problems (general case and specific ones discussed in this paper). This is important in order to avoid extensive simulations, to extend results to systems with offset, and most likely it will be necessary to set up formal proves of the conjectures. Finally we would like to explore implementation issues of a dual priority scheduler in the HIPPEROS real-time operating system [17].

Acknowledgments

We thank the anonymous reviewers for their constructive comments.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese, "Polynomial-time exact schedulability tests for harmonic real-time tasks," in *34th IEEE Real-Time Systems Symposium*, December 2013, pp. 236–245.

[3] A. Burns and A. Wellings, "Dual priority assignment: A practical method for increasing processor utilisation," in *Fifth Euromicro Workshop on Real-time Systems*, Oulu, Finland, June 1993, pp. 48–53.

[4] L. George, J. Goossens, and D. Masson, "Dual Priority and EDF: a closer look," in *Proceedings of the Work-in-Progress Session of 35th IEEE Real-Time Systems Symposium (RTSS 2014 WiP)*, Rome, Italy, Dec. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01217433

[5] R. M. Pathan, "Unifying fixed and dynamic-priority scheduling based on priority promotion and improved ready queue management technique," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 209–220.

[6] R. Davis and A. Wellings, "Dual priority scheduling," in *16th IEEE Real-Time Systems Symposium*, Pisa, Italy, 1995, pp. 100–109.

[7] R. Davis, "Dual priority: a means of providing flexibility in hard real-time systems," University of York, England, Tech. Rep. YCS-94-230, 1994.

[8] J. M. Banús, A. Arenas, and J. Labarta, "Extended global dual priority algorithm for multiprocessor scheduling in hard real-time systems," in *Euromicro Conference on Real-Time Systems WIP (ECRTS), Palma de Mallorca, Spain*, 2005, pp. 13–16.

[9] A. Burns, "Dual priority scheduling: Is the processor utilisation bound 100%?" in *1st International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, Brussels, Belgium, 2010.

[10] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Department of Computer Science, University of York, Tech. Rep., 1991.

[11] S. Han and M. Park, "Predictability of least laxity first scheduling algorithm on multiprocessor real-time systems," in *EUC workshops*. Springer, 2006, pp. 755–764.

[12] M. Joseph and P. Pandya, "Finding response times in a real-time system," in *The Computer Journal*, 1986, p. 29(5):390–395.

[13] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2017, pp. 139–146.

[14] Rt simulator. [Online]. Available: https://github.com/paul-rodriguez/rt

[15] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS workshop at the Euromicro Conference on Real-Time Systems*, 7 2010, pp. 6–11.

[16] J. Goossens and C. Macq, "Limitation of the hyper-period in real-time periodic task set generation," in *In Proceedings of the RTS Embedded System (RTS'01*, 2001, pp. 133–147.

[17] A. Paolillo, O. Desenfans, V. Svoboda, J. Goossens, and B. Rodriguez, "A new configurable and parallel embedded real-time micro-kernel for multi-core platforms," *OSPERT 2015*, p. 25, 2015.