Non-Work-Conserving Scheduling of Non-Preemptive Hard Real-Time Tasks Based on Fixed Priorities

Mitra Nasri, Gerhard Fohler
Chair of Real-time Systems,
Technische Universität Kaiserslautern, Germany
{nasri, fohler}@eit.uni-kl.de

ABSTRACT

In this paper, we consider scheduling of non-preemptive and independent periodic tasks with implicit deadlines on uniprocessor systems. We assume **loose-harmonic** task sets with periods being integer multiples of the smallest period. This problem is known to be NP-Hard. Recently, two online non-work-conserving scheduling algorithms called Precautious-RM (P-RM) and lazy P-RM (LP-RM) have been introduced to guarantee schedulability of special cases of *harmonic* task sets. In this paper, we provide new sufficient schedulability tests for P-RM and LP-RM. Then, by introducing a new algorithm called efficient Precautious-RM (EP-RM), we increase the number of schedulable loose-harmonic task sets in comparison with P-RM by letting the tasks share the priorities. We prove that the new test dominates the schedulability tests for P-RM. Moreover, we show that if the ratio of two consecutive periods is greater than or equal to 3, P-RM, LP-RM, and EP-RM are able to schedule any task set satisfying the necessary conditions of schedulability. We present results from a simulation study, showing the efficiency of the algorithm.

1. INTRODUCTION

In many real-time embedded or networked systems preemption is either not possible or prohibitively expensive [10]. For example, in controller area networks (CAN), messages are transmitted non-preemptively [20]. From the application's perspective, non-preemptive execution reduces the delay between I/O operations, which is an important concern in the design of control systems [4, 19]. From the system's perspective, it helps to have simpler mechanisms for handling mutual exclusion. It facilitates the precision of the worst-case execution time (WCET) estimation [18]. Because of these reasons, non-preemptive execution has been used in many avionics, robotics, and mobile communication applications [17, 1].

Non-preemptive scheduling of independent real-time periodic tasks with deadline equal to period on a single processor system is NP-Hard [11], even if periods are harmonic [6] or powers of 2 [16]. A necessary and sufficient test for schedulability of non-preemptive EDF (npEDF) for general periodic tasks with arbitrary release offsets has been presented in [11]. Later, sufficient conditions for the schedulability of periodic and sporadic tasks scheduled by non-preemptive rate-monotonic (npRM) have been studied in [17, 2] and [9], respectively. For harmonic task sets, another sufficient schedulability test for npEDF (and npRM) has been presented in [14]. A schedulability test for strictly periodic tasks, i.e., task sets with no start time jitter,has been presented in [13]. However, as mentioned in [14], because of the work-conserving nature of npEDF and npRM, they might not be able to schedule many of the feasible task sets.

In [8] and [12], two heuristic non-preemptive scheduling algorithms have been introduced. Clairvoyant EDF (cEDF) [8], which is for firm real-time tasks, uses a form of look-ahead to determine the appropriate time to insert an idle interval before starting the execution of an urgent task. For that it may need to search all jobs in a hyperperiod. The second algorithm which is called group-based EDF (gEDF) [12], is an online scheduling algorithm for soft real-time tasks. It groups the tasks based on the closeness of their deadlines, and then schedules a task with the smallest execution time among the tasks of the group with the earliest deadline. However, these heuristics cannot guarantee hard deadlines.

Since the problem of non-preemptive scheduling is NP-Hard, solutions for special cases, such as harmonic task sets, have been presented. In [7], a non-preemptive scheduling algorithm has been introduced for task sets with constant period ratio K, i.e., each period must be exactly $K \in \mathbb{N}$ times greater than the period of the task with the next-smallest period, for some integer constant $K \geq 3$. In [6], this algorithm is extended to guarantee schedulability of $K = 2$. Besides, it considers another case where tasks can have different period ratio which must be an integer greater than 2. Computational complexity of the algorithms presented in [7, 6] is exponential.

Recently, two online non-preemptive non-work-conserving scheduling algorithms called Precautious-RM (P-RM) and Lazy-Precautious-RM (LP-RM) have been introduced [15] for harmonic tasks. P-RM guarantees the schedulability of the task sets in [7] and [6]. Both of these algorithms can successfully schedule a set of harmonic tasks with arbitrary integer period ratio as long as there are enough *vacant intervals* in the task set. Vacant intervals are counted as the number of pairs of instances of the task that has the smallest period. P-RM and LP-RM are efficient online scheduling algorithms

RTNS 2015, November 04 - 06, 2015, Lille, France

 ISBN 978-1-4503-3591-1/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2834848.2834856

because if the system supports one priority queue for each priority level, their computational complexity is $O(1)$.

Motivating by industrial applications such as [3], in this paper we consider implicit deadline tasks with periods which are integer multiples of the smallest period. We call such task set as *loose-harmonic*. The contributions of the paper can be summarized as

- Providing linear-time sufficient schedulability tests for P-RM and LP-RM in loose-harmonic task sets.
- Introducing Efficient Precautious-RM (EP-RM) which handles multiple tasks in priority groups to increase the number of schedulable task sets. Tasks which are assigned to a priority group share one ready queue and are allowed to be scheduled only if the task with the smallest period among them is scheduled in the current vacant interval. Because of letting tasks share one vacant interval, EP-RM can admit more feasible task sets than P-RM.
- Providing a linear-time schedulability test for EP-RM.
- Presenting an offline heuristic algorithm (with polynomial time computational complexity) to assign tasks to priority groups and showing that this algorithm theoretically dominates the schedulability conditions of P-RM.
- Showing that if the period ratio is greater than or equal to 3, a feasible loose-harmonic task set is always schedulable by LP-RM, P-RM and EP-RM.

The rest of the paper is organized as follows; in Sect. 2 we introduce the system model and background including P-RM and LP-RM algorithms. In Sect. 3 we present a sufficient schedulability test for these algorithms in loose-harmonic task sets. Then in Sect. 4 we present EP-RM algorithm together with its schedulability tests and a priority assignment algorithm. We evaluate the algorithms in Sect. 5 and conclude the paper in Sect. 6.

2. SYSTEM MODEL AND BACKGROUND

We consider a set of independent, non-preemptive periodic tasks, with deadlines equal to periods and no release offset on a single processor. Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ has n tasks and each task is identified by $\tau_i : (c_i, T_i)$, where $c_i \in \mathbb{N}^+$ is the worst-case execution time, and $T_i \in \mathbb{N}^+$ is the period. We assume c_i and T_i are integer multiples of the system's clock, thus they are integer values. Tasks are periodic, have no release offset, and have implicit deadlines. We denote system utilization by $U = \sum_{i=1}^{n} u_i$ where $u_i = c_i/T_i$ is the utilization of task τ_i. The hyperperiod is denoted by H and is the least common multiple of the periods. Tasks are indexed according to their period so that $T_1 \leq T_2 \leq \ldots \leq T_n$. The period ratio of task τ_i $(1 < i \leq n)$ is defined as $k_i = T_i/T_{i-1}$, $k_i \in \mathbb{R}$, representing the ratio of two consecutive periods. We call the task set *loose-harmonic* because we assume that $\forall i, T_i/T_1 \in \mathbb{N}$. In other words, releases of the low priority tasks is synchronous with the highest priority task, i.e., τ_1.

The work presented in this paper is based on P-RM [15] which is an online non-preemptive scheduling algorithm. It takes actions at the following events; a) finishing of an instance of a task (called a job), b) finishing of a scheduled idle interval, and c) a job release when no other task is executing. For the sake of simplicity, we assume that the scheduler is not activated until the execution time of the task is completely finished.

Alg. 1 presents P-RM. Similar to RM, P-RM prioritizes the tasks based on their period, i.e., τ_1 and τ_n are the tasks with the highest and the lowest priorities. In P-RM, one of the following conditions must hold to be able to schedule a low priority task τ_i: a) the task must be finished before the next release of τ_1, or b) the previously executed task must be τ_1 and the current task must be able to finish its execution before the latest time at which the next instance of τ_1 can start and finish its execution. This instant is $r_1^{next} + (T_1 - c_1)$. If any of these conditions are not satisfied for τ_i, P-RM schedules an idle interval until the next release of τ_1 at r_1^{next} (Lines 8 and 9).

Algorithm 1 Precautious-RM

Input t: t is the current time
Output S, t^{next}: S is the next task to schedule and t^{next} is the next time to activate the scheduler.

1: $r_1^{next} \leftarrow \Big(\lfloor t/T_1 \rfloor + 1\Big)T_1$
2: $i \leftarrow$ index of the highest priority task with a pending job
3: **if** $((t + c_i \leq r_1^{next})$ **or** $(t + c_i \leq r_1^{next} + T_1 - c_1$ **and** τ_1 is the latest executed task)) **then**
4: $\quad S \leftarrow \tau_i$
5: $\quad t^{next} \leftarrow t + c_i$
6: $\quad$ **return** S, t^{next}
7: **else**
8: $\quad S \leftarrow null$
9: $\quad t^{next} \leftarrow r_1^{next}$
10: **end if**
11: **return** S, t^{next}

In Alg. 1, schedulability of τ_1 is always guaranteed because due to Line 3, no low priority task is allowed to start its execution if it will cause a deadline miss for the next instance of τ_1. Thus, if the current instance of τ_1 has not yet been scheduled, t, which is the time that the scheduler is activated, is smaller than $r_1^{next} - c_1$. Consequently, if $i = 1$, the first condition in Line 3 becomes true.

In general, the computational complexity of P-RM is $O(n)$ because of Line 2. However, similar to the RM algorithm, there can be an efficient implementation for P-RM in systems which support one ready queue for each priority level. In that case, the computational complexity is $O(1)$. As mentioned in [5], if the number of priority levels is not high, finding the highest priority task with a pending job can be implemented more efficiently by splitting the ready queue into several FIFO queues, one for each priority level. Thus, whenever a pending job is inserted into a ready queue, we can set its corresponding flag to 1. Then, it is possible to use a widely implemented CPU instruction to find the index of the most significant bit in a given integer value. The result is the index of a non-empty ready queue with the highest priority.

Another variant of P-RM is LP-RM [15]. LP-RM schedules any task with lower priority than τ_1, between two instances of τ_1. From the implementation point of view, the only difference between these two algorithms is in the con-

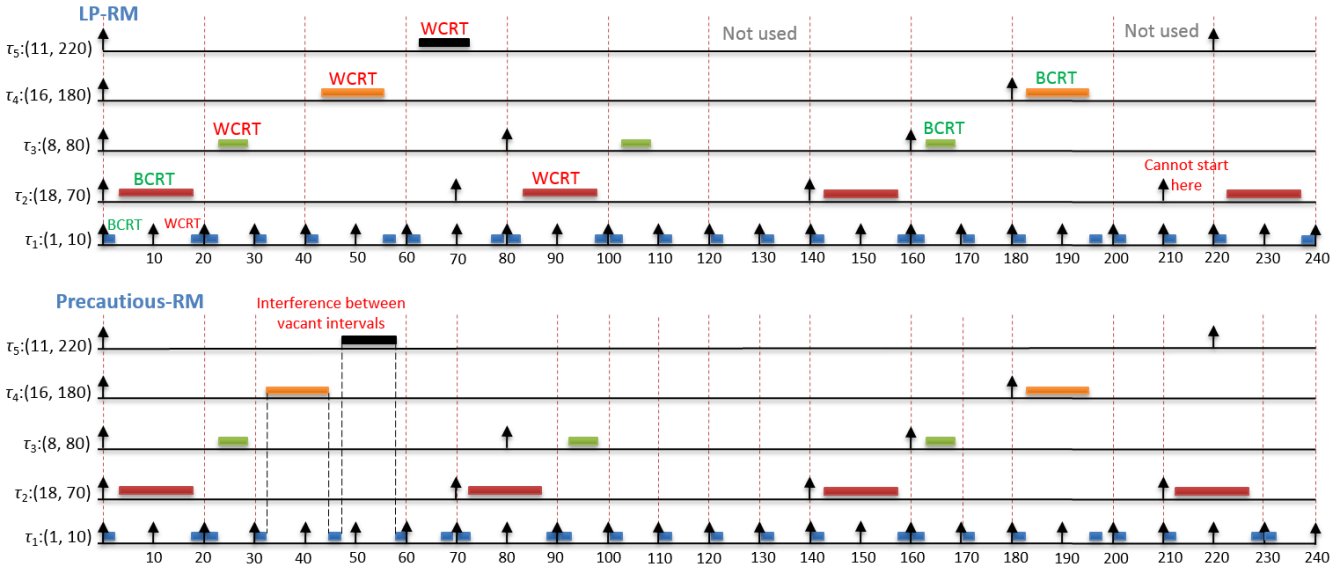

Figure 1: P-RM and LP-RM schedules for an example task set with 5 tasks. Note that this task set cannot be feasibly scheduled by npRM or npEDF because of the execution of τ_4 right after τ_3 at time 29 which leads to deadline miss of τ_1 at time 40. WCRT and BCRT are the worst- and best-case response times of the tasks.

dition of Line 3 of Alg. 1. In LP-RM, Line 3 of Alg. 1 is replaced by the condition in Alg. 2.

Algorithm 2 LP-RM

The following if-statement replaces Line 3 of Alg. 1. The rest is identical with Alg. 1.

if $((i = 1)$ **or** (τ_1 is the latest executed task **and** $f(t) = 0$ **and** $t + c_i \leq r_1^{next} + T_1 - c_1)$) **then**

In Alg. 2, $f(t)$ is an indicator to show whether the number of previously released jobs of τ_1 is even or odd. Namely, $f(t) = 0$ if $\lfloor t/T_1 \rfloor = 2h$ for $h \in \mathbb{N}$, otherwise $f(t) = 1$. Figure 1 shows two schedules produced by LP-RM and P-RM for the same task set.

In this paper, a job of task τ_1 is called an *even job* if it is released at time t where $\lfloor t/T_1 \rfloor = 2h$ for $h \in \mathbb{N}$. Otherwise, the job is called *odd job*.

3. A SCHEDULABILITY TEST FOR P-RM IN LOOSE-HARMONIC TASK SETS

In this section, first we present a sufficient schedulability test for LP-RM and then extend and customize it for P-RM. Our idea is similar to the one used in [15] and [14]; we count the number of *vacant intervals* for each task and check if there are enough vacant intervals to schedule all of the tasks. In [15], a vacant interval is defined as:

Definition 1. *A vacant interval is a time interval with length $2(T_1 - c_1)$ which can be constructed between two consecutive instances of τ_1 (when one of them is scheduled at its release, and the other one is possibly scheduled at the end of its period).*

A vacant interval can be used for scheduling low priority tasks in the system. According to [15], in a schedule produced by LP-RM for *harmonic* task sets, the following equation determines the number of vacant intervals available to schedule lower priority tasks than τ_i ($\forall i, 1 < i \leq n$):

$$v_i = k_i v_{i-1} - 1 \tag{1}$$

where $v_1 = 0.5$ and k_i is the period ratio (see Sect. 2). Although the concept of vacant interval defined in Definition 1, needs two consecutive instances of τ_1, we mathematically assume this definition can be extended to τ_1 by considering $v_1 = 0.5$. In other words, τ_1 has half of the slack of a vacant interval. In this equation, v_{i-1} shows the number of vacant intervals in one release of τ_{i-1}. Since due to the harmonic assumption in [15], each period is exactly $k_i \in \mathbb{N}$ times greater than T_{i-1}, the number of available vacant intervals before scheduling τ_i is $k_i v_{i-1}$. To schedule τ_i, a vacant interval is needed, thus, we must decrease $k_i v_{i-1}$ by 1 which leads to $v_i = k_i v_{i-1} - 1$. In the appendix we provide necessary and sufficient conditions of schedulability of LP-RM when it is used to schedule harmonic task sets.

Extending the notion of counting vacant intervals in [15], in this paper we obtain a lower bound for v_i such that for any possible release sequence of the high priority tasks, v_i vacant intervals are guaranteed to be available for other tasks with lower priority than τ_i per every release of τ_i. Accordingly, even if a low priority task τ_j ($i < j$) is executing at the release of τ_i, v_i is still the same.

Theorem 1. *Given a task set τ defined in Sect. 2 scheduled by LP-RM, there will be at least v_i vacant intervals available for the low priority tasks τ_j ($i < j \leq n$) during every release of τ_i, with v_i calculated as follows*

$$v_i = \lfloor k_i \rfloor v_{i-1} - 1 \tag{2}$$

and $v_1 = 0.5$.

Proof. We use induction for the proof. *Induction basis:* During each arbitrary time interval $[t, t+T_2]$ where $t \in [0, H]$ (and $t/T_1 \in \mathbb{N}$), there will be at least v_2 vacant intervals

where v_2 is obtained from (2), and H is the hyperperiod. The proof is straight forward because all releases of τ_2 are synchronous with the releases of τ_1. Thus, we always have $k_2 \in \mathbb{N}$ number of releases of τ_1 within one release of τ_2. In other words, the number of releases of τ_1, and the amount of vacant intervals for τ_2 are always the same for every job of τ_2. Since τ_2 takes one of those vacant intervals, and each vacant interval is created from one pair of τ_1 jobs, the remaining vacant intervals follow (2).

Induction assumption: During each release of τ_{i-1} at any time $t \in [0, H]$ ($t/T_1 \in \mathbb{N}$) there will be v_{i-1} vacant intervals obtained from (2) in interval $[t, t + T_{i-1}]$. *Induction hypothesis:* During each release of τ_i at any time $t \in [0, H]$ ($t/T_1 \in \mathbb{N}$) there will be v_i vacant intervals obtained from (2) in interval $[t, t+T_i]$. According to the induction assumption, for every value of t, we will have v_{i-1} vacant intervals during the interval $[t, t+T_{i-1}]$. Thus, for every possible release time of τ_i at an arbitrary time instant such as t, we will have v_{i-1} vacant intervals. Since $T_{i-1} \leq T_i$, there will be at least $\lfloor k_i \rfloor$ releases of τ_{i-1} during one release of τ_i. Using the induction assumption, we can assume that the first such release will be at t, synchronous with the release of τ_i, and the other $\lfloor k_i \rfloor - 1$ releases will appear periodically after t. Since they have started from t, the deadline of the latest released τ_{i-1} will be at $\lfloor k_i \rfloor T_{i-1}$ which is smaller than or equal to $t + T_i$. Since v_{i-1} vacant intervals are guaranteed for each of those releases and τ_i is scheduled in one of them, the remaining number of vacant intervals is at least $\lfloor k_i \rfloor v_{i-1} - 1$ which follows formula (2). And hence, the theorem is proven. □

In the next step, we derive schedulability conditions for LP-RM for our task model in Sect. 2. Due to the asynchronous releases of the tasks, if any of the jobs of a task such as τ_i, $1 < i \leq n$ are released together with an odd job of τ_1, it will be exactly in the middle of a vacant interval. Since LP-RM does not allow a low priority task to start its execution in the middle of a vacant interval, i.e., when $f(t) = 1$, we must make sure that v_i is greater than or equal to 0.5, otherwise, τ_i may not be able to fit into one of the vacant intervals. However, if all releases of τ_i are synchronous with the start of the vacant intervals, i.e., $T_i/T_1 = 2h$, for $h \in \mathbb{N}$, the processor will never be busy with the execution of a task during the times when τ_i is released. For those cases, we must only check if v_i is greater than 0 (for $i \neq n$). The following theorem presents our set of sufficient conditions for the schedulability of LP-RM.

THEOREM 2. *Given task set τ according to Sect. 2, the following conditions provide a sufficient schedulability test for LP-RM*

$$\sum_{i=1}^{n} u_i \leq 1 \tag{3}$$

and $\forall i, 1 < i \leq n$:

$$c_i \leq 2(T_1 - c_1), \tag{4}$$

$$v_i \geq \begin{cases} 0, & i = n \textbf{ and } T_n/T_1 \textit{ is even} \\ 0.5, & \textit{otherwise} \end{cases} \tag{5}$$

PROOF. Necessity of conditions (3) and (4) has already been proven in [11] and [6], respectively. As shown by Theorem 1, every release of task τ_i, has at least v_i units of vacant intervals which are available for other lower priority tasks. Also in the worst-case, jobs of the tasks may release in the middle of a vacant interval where $f(t) = 1$, hence, they must have at least 0.5 units of vacant interval to tolerate such blocking.

We show that if (5) is satisfied, each job of a task has its guaranteed slot even if it is released when $f(t) = 1$. Since according to the condition of Alg. 2, a low priority task such as τ_x ($i < x \leq n$) starts its execution at time t only if it can be finished before $r_1^{next} + T_1 - c_1$, it is guaranteed that at r_1^{next}, where τ_i is released, the processor will be busy with the execution of τ_x at most for half a vacant interval. Note that if τ_i is already released at t, τ_x has no chance to start its execution because our algorithm applies RM priorities. Consequently, the maximum blocking time for a task is 0.5 vacant intervals. According to equation (1), v_i is calculated in the way that the execution slot for τ_i is guaranteed. Moreover, due to (5), each task has at least 0.5 vacant intervals which can be used to compensate its blocking time by other low priority tasks. As a result, if all tasks τ_j, $1 < j < n$ have at least 0.5 units of v_j, all of them can tolerate such blocking while they can be executed in their guaranteed slot. Task τ_n is an exception because there will be no lower priority task than τ_n to block the processor at the release of τ_n. Thus, if T_n/T_1 is even, there is no need to keep 0.5 vacant intervals for v_n. □

Note that if $v_i = 0$ for $i \neq n$, there might not be any vacant interval in one release of τ_i, and hence, τ_{i+1} will have a negative value of v_{i+1} because $v_{i+1} = \lfloor k_{i+1} \rfloor v_i - 1$. Formula (5) also guarantees the existence of the free vacant intervals to schedule τ_i.

As the next contribution of this paper, we extend Theorem 2 to present a sufficient schedulability test for P-RM. According to the example in Figure 1, P-RM is able to schedule two or more low priority tasks in 3 releases of τ_1, as it has happened from time 30 to 60 for the first jobs of τ_4 and τ_5. In other words, P-RM lets tasks *share* their own vacant intervals. Moreover, it does not necessarily schedule low priority tasks between even jobs of τ_1, as it happened for the second instance of τ_3 at 101. To derive schedulability conditions, first, we define the owner of a vacant interval.

DEFINITION 2. *The owner of a vacant interval which is used to schedule a non-empty sequence of low priority tasks that are executed after one instance of τ_1 in the vacant interval, is the first low priority task which is executed after the instance of τ_1 in that vacant interval.*

For example, in Figure 2-a, the owner of the vacant interval $[t, t + T_1]$ is τ_2. Note that since $c_2 < T_1 - c_1$, τ_2 cannot own more than half a vacant interval. If the second half of the vacant interval is free (as it happened at time $t + T_1$), a new vacant interval starts afterward. In Figure 2-b, the owner of a vacant interval from $[t - T_1, t + T_1]$ is τ_x because we have $c_x > T_1 - c_1$. Thus, τ_x owns the whole vacant interval. Note that even though τ_2 is started at the second half of the vacant interval it is not the owner of the second half. Instead, τ_2 owns the first half of the vacant interval which has been started from $t + T_1$.

LEMMA 1. *If a task such as τ_i ($1 < i \leq n$) has $c_i \leq T_1 - c_1$, it never owns more than half of a vacant interval. Otherwise, if $c_i > T_1 - c_1$ in a feasible schedule, τ_i may own one full vacant interval.*

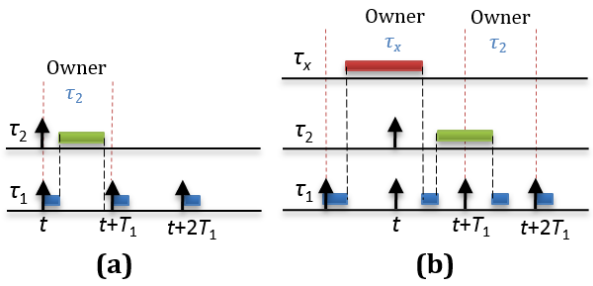

Figure 2: An example for P-RM schedule when vacant intervals may merge with each other

PROOF. As shown in Figure 2-a, if the task is scheduled at the start of a free vacant interval, it owns half of the vacant interval because of the fact that $c_1 + c_i < T_1$. Otherwise, if the task is started in a vacant interval which is owned by another task, and is finished during the next release of τ_1, as shown in Figure 2-b, it owns half of a vacant interval because $\sigma_i + c_1 < T_1$ where σ_i is the remaining execution time of τ_i at the release time of the next τ_1. The proof for the second case follows directly from the fact that in a feasible schedule, $c_i \leq 2(T_1 - c_1)$ [6], thus, τ_i cannot occupy more than one full vacant interval. □

In the next theorem we provide an equation to obtain the lower bound of available vacant intervals in a P-RM schedule.

THEOREM 3. *Given task set τ defined in Sect. 2 scheduled by P-RM, there will be at least*

$$v_i = \begin{cases} \lfloor k_i \rfloor v_{i-1} - 0.5, & c_i \leq T_1 - c_1 \ \textbf{and}\ 1 < i \leq n \\ \lfloor k_i \rfloor v_{i-1} - 1, & c_i > T_1 - c_1 \ \textbf{and}\ 1 < i \leq n \end{cases} \quad (6)$$

vacant intervals after any job release of task τ_i $(1 < i \leq n)$, where $v_1 = 0.5$. These vacant intervals are available for low priority tasks τ_j, $(i < j \leq n)$.

PROOF. The proof is similar to the proof of Theorem 1 considering that instead of 1 unit of v_i for each task, if $c_i \leq T_1 - c_1$, according to Lemma 1, we only need to decrease 0.5 units from the available vacant intervals. □

As shown in Figure 1 and 2, regardless of the value of T_i/T_1, any of the tasks with higher priority than τ_i may be released while the processor is busy with the execution of a low priority task. Thus, in our schedulability test, we force them to have at least 0.5 units of a vacant interval for such situation.

THEOREM 4. *Given task set τ in Sect. 2 is schedulable by P-RM if the following set of conditions hold: (3), (4), $v_n \geq 0$, and $\forall i, 1 < i < n$ we have*

$$v_i \geq 0.5 \quad (7)$$

PROOF. The proof is the same as the proof of Theorem 2. The only difference is that in P-RM, all tasks may release when the processor is busy with the execution of a low priority task. Thus, regardless of T_i/T_1 being even or odd, 0.5 of a vacant interval of a task may be used by a low priority task. However, schedulability is guaranteed since each task has its guaranteed vacant interval (according to (6)). In addition, according to condition (7), each task τ_i $(1 < i < n)$ has extra 0.5 units of vacant interval to compensate the blocking which is caused by the low priority tasks. Finally, since in P-RM, the start times of the tasks are not restricted to $f(t) = 0$, task τ_n does not need 0.5 units of v_n to compensate for the blocking. □

4. IMPROVED SCHEDULABILITY USING EP-RM

In this section, first we introduce EP-RM and priority groups. Then we present a set of heuristic algorithms for assigning tasks to the priority groups. Finally we discuss theoretical properties of EP-RM, such as the dominance of our schedulability test in EP-RM in comparison with P-RM. Moreover, we discuss the computational complexity of the tests and the priority assignment algorithms.

4.1 Efficient Precautious-RM

The goal of this section is to increase the number of schedulable tasks by letting them have the same priority. We modify P-RM so that instead of one task, it schedules all other tasks with the same priority and pending jobs inside one vacant interval. Each priority group is denoted by $P_i = \{\tau_1^i, \tau_2^i, \ldots, \tau_{X_i}^i\}$ $(1 \leq i \leq m)$ where i indicates the priority level, m is the number of priority levels, and X_i is the number of tasks in P_i. We assume that the tasks inside a priority group are sorted according to their period so that $T_1^i \leq T_2^i \leq \ldots \leq T_{X_i}^i$. Each $\tau_j^i \in \tau$ $(1 \leq j \leq X_i)$ represents one task in the task set. Every priority group has a *representative task*, denoted by τ_1^i. This task has the smallest period among the other periods, thus, it has the highest priority under rate monotonic. The other tasks in the priority group are called *tail tasks* because they are allowed to be scheduled only if an instance of the representative task is scheduled in the same vacant interval.

If some tasks are grouped into one priority group, they do not consume more than one vacant interval, and thus, during the schedulability analysis, we need less vacant intervals for lower priority tasks. However, it is very important to note that if we do not force the tail tasks to be scheduled only after the representative task, as shown in the example in Figure 3, they might take their own vacant intervals in future releases. In that case, in the worst-case behavior, we must consider that any of them occupies its own vacant interval, and hence, the schedulability is not efficiently improved. This situation is caused by asynchronous releases of the tasks. For example, in Figure 3 when the second instance of τ_3 is scheduled, the second instance of τ_5 is not released, thus, some parts of the vacant interval for τ_3 are wasted, and at time 90 when τ_5 is released, it takes another vacant interval for itself.

As mentioned earlier, to solve the previous issue, we force tail tasks to execute only after the start of the schedule of τ_1^i. Thus, even if these tasks are released, they must wait until P-RM schedules the representative task. Then, among the tasks in the group, those which are not yet executed are scheduled after τ_1^i. Moreover, we assume we will not schedule two instances of the same task in one vacant interval. To distinguish between the two versions of P-RM, we call the second one which is introduced in this section, efficient Precautious-RM. The following theorem provides sufficient conditions for the schedulability of the tail tasks.

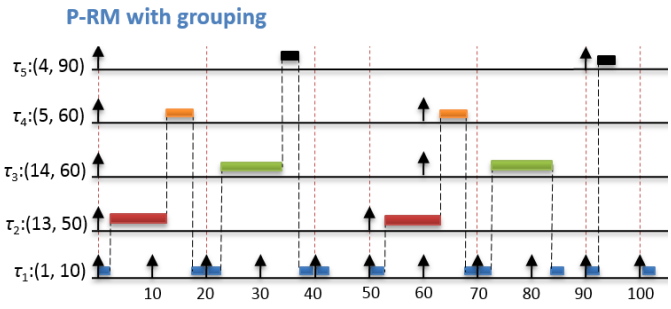

Figure 3: An example for the issue regarding task grouping in P-RM. In this example, τ_2 and τ_4 are in a one group and τ_3 and τ_5 are in another group.

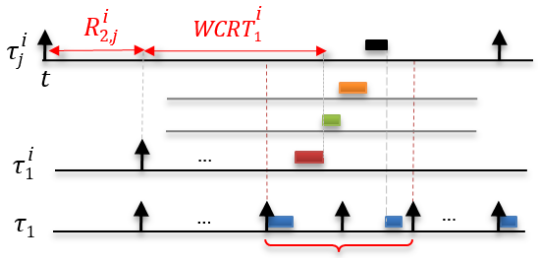

Figure 4: A symbolic example to show the worst-case activation scenario for a tail task

THEOREM 5. *Given priority group P_i, if the representative task τ^i_1 can be successfully scheduled by EP-RM, all tail tasks can also be scheduled successfully if $C_i \leq 2(T_1 - c_1)$ and $\forall j, 1 < j \leq X_i$, we have*

$$R^i_{1,j} + WCRT^i_1 + \sum_{x=2}^{X_i} c^i_x \leq T^i_j \tag{8}$$

where $WCRT^i_1$ is the worst-case response time of τ^i_1, $R^i_{1,j}$ is the maximum distance between any release of a job of τ^i_1 from the latest released job of τ^i_j during the hyperperiod, and $C_i = \sum_{j=1}^{X_i} c^i_j$.

PROOF. If condition $C_i \leq 2(T_1 - c_1)$ holds, all tasks in the priority group can be scheduled in at most one vacant interval. According to the assumption of theorem, the schedulability of τ^i_1 is already guaranteed, thus, whenever it is scheduled, other released tail tasks can also be scheduled. Now we must show that if (8) holds, none of the tail tasks miss their deadlines. As shown in Figure 4, in the worst-case, τ^i_j is released at time t after the execution of τ^i_1. Hence, it must wait for the next release of τ^i_1. In the worst-case, that release happens $R^i_{1,j}$ units of time after t. Also in the worst-case, τ^i_1 will have its worst-case response time in that release, thus, it will be finished at $R^i_{1,j} + WCRT^i_1$ units of time after t. Now, assume that in the worst-case, all of the tail tasks with lower index than τ^i_j are waiting for their execution. Consequently, if the finish time of τ^i_j at $R^i_{1,j} + WCRT^i_1 + \sum_{x=2}^{j} c^i_x$ is still smaller than its deadline, it can be successfully scheduled by EP-RM. However, to cope with the situations where T^i_j is in the middle of the vacant interval in which τ^i_1 is executing, we assume $\sum_{x=2}^{X_i} c^i_x$ instead of $\sum_{x=2}^{j} c^i_x$. □

The next step is to derive schedulability conditions for the representative tasks. For this, we revise the way we calculate the vacant intervals. We use V_i to represent the available vacant intervals for priority groups with lower priority than P_i as

$$V_i = \begin{cases} \lfloor K_i \rfloor V_{i-1} - 0.5, & C_i \leq T_1 - c_1 \\ \lfloor K_i \rfloor V_{i-1} - 1, & C_i > T_1 - c_1 \end{cases} \tag{9}$$

where $V_1 = 0.5$ and K_i is the period ratio of two priority groups defined as

$$K_i = \frac{T^i_1}{T^{i-1}_1} \tag{10}$$

The following theorem concludes sufficient schedulability test for EP-RM. We assume that the only task in P_1 is $\{\tau_1\}$, thus, it has $V_1 = 0.5$. If P_1 has more than one task, the length of each vacant interval must be calculated as $T_1 - C_1$ where C_1 is sum of the execution times of the tasks assigned to P_1. Since this greatly affects the schedulability condition (4), we simply assume that P_1 only contains τ_1.

THEOREM 6. *Task set τ with m priority groups P_1 to P_m can be successfully scheduled by EP-RM if $\forall i, 1 < i \leq m$, we have $C_i \leq 2(T_1 - c_1)$, condition (8), $V_m \geq 0$ and $\forall i, 1 < i < m$ we have*

$$V_i \geq 0.5 \tag{11}$$

PROOF. The proof is based on the proof of Theorem 4. Instead of having n tasks in the task set, we have m tasks; each is represented by a priority group. Since no priority group will occupy more than one vacant interval (due to $C_i \leq 2(T_1 - c_1)$), and will never produce more scheduling requests than τ^i_1 which has the smallest period among the tasks in the group, the representative task can be a representative for all other tasks in the group. Thus, we can simply imagine a new task set $\tau' = \{\tau'_1, \tau'_2, \ldots, \tau'_m\}$ where $\tau'_i = (C_i, T^i_1)$. Because of (11) and Theorem 4 we know that the schedulability of the representative task is guaranteed. Thus, sufficient schedulability conditions of Theorem 5 for the tail tasks are also satisfied. □

Having the schedulability conditions for EP-RM, we only need to assign tasks into priority groups so that $C_i \leq 2(T_1 - c_1)$ and (8) holds. For the latter, we need to find the WCRT of the representative task, and the worst possible release sequence. Since it might be hard to calculate the WCRT in non-harmonic task sets, we devise a simpler test, which can be verified in $O(1)$ to guarantee (8) for a priority group.

THEOREM 7. *Task set τ which is partitioned into m priority groups P_1 to P_m can be successfully scheduled by EP-RM if $\forall i, 1 < i \leq m$, $C_i \leq 2(T_1 - c_1)$, condition (11) and*

$$T^i_2 \geq 2T^i_1 \tag{12}$$

are satisfied.

PROOF. As shown in Figure 5, if (12) holds, during one release of every job of τ^i_2 at time t, there will be at least 2 releases of τ^i_1. Thus, even if τ^i_2 cannot be scheduled until the deadline of an instance of τ^i_1 which has been released before t (we call it the first instance), it can be scheduled after the schedule of the second instance of τ^i_1. Since the third instance of τ^i_1 will also be released before the deadline

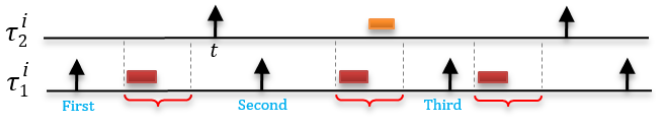

Figure 5: A symbolic example to show that if $T_2^i \geq 2T_1^i$, (8) is satisfied

of τ_2^i, it is guaranteed that τ_2^i can be finished before the second instance of τ_1^i, i.e., before its own deadline. Since we have assumed that tasks are indexed according to their periods, if $T_2^i \geq 2T_1^i$, all other tasks τ_j^i $(2 < j \leq X_i)$ in P_i will have $T_j^i \geq T_2^i \geq 2T_1^i$, and hence, the theorem is proven because all of the schedulability conditions of Theorem 6 are satisfied. □

4.2 Assigning Tasks to Priority Groups

The problem of assigning tasks to priority groups can be seen as a bin-packing problem with some additional conditions; a) each priority group has a capacity of $2(T_1 - c_1)$, b) each task τ_j can be assigned to P_i if $T_j \geq 2T_1^i$ according to Theorem 7, c) the number of the groups is not known in advance, d) V_i is a function of the period ratio of the representative tasks, i.e., $\lfloor K_i \rfloor$, thus, the choice of the representative tasks affects the schedulability, and e) condition (11) must be satisfied. It is worth mentioning that even without these conditions, the basic bin-packing problem is NP-Complete.

In the next step we provide a heuristic task assignment algorithm based on the *first-fit* policy. We try to fit each task to one of the existing groups. If not possible, we add another group for that task. The first group which can accept the task would be its hosting group. However, because of our schedulability conditions, we cannot merely consider the execution times of the tasks. If by adding a task such as τ_i to a group such as P_j, its C_j becomes greater than $T_1 - c_1$, instead of 0.5 units, it takes 1 unit of V_j. Hence, in some cases, for example if the period ratio is 2, it might be better to leave the group half-empty (with $C_j \leq T_1 - c_1$) instead of creating a group with bigger size. Having a group with $C_j > T_1 - c_1$, affects V_x of other priority groups $(j < x \leq m)$, and may make them unschedulable. Assume that after adding τ_i into P_j, the resulting priority group is P'. Task τ_i can fit to the group if for P' we have

$$c_i + C_j \leq 2(T_1 - c_1) \textbf{ and } T_i \geq 2T_1^j \textbf{ and } \text{(11) holds for } P' \quad (13)$$

Condition (13) is necessary for the schedulability of the first-fit algorithm according to Theorem 7. Based on this condition, we can implement the next-fit and best-fit algorithms as well. It is worth mentioning that if $c_i + C_j \leq T_1 - c_1$, adding τ_i to P_j only improves the schedulability in comparison with P-RM. The reason is that according to (6), in P-RM we waste 0.5 units of vacant intervals from every task which is currently assigned to P_j plus τ_i itself. This amount is greater than or equal to $(X_j + 1) \times 0.5$, while in EP-RM, we only need 0.5 units of vacant intervals which is taken from τ_1^j.

Although condition (13) does not threaten the schedulability of the existing priority groups, it cannot guarantee the schedulability of the future groups.

EXAMPLE 1. *Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_6\}$ with $\tau_1 = (3, 10)$, $\tau_2 = (2, 20)$, $\tau_3 = (1, 40)$, $\tau_4 = (5, 70)$, $\tau_5 = (12, 130)$, and $\tau_6 = (7, 330)$ does not have the schedulability conditions of EP-RM (Theorem 7) if the first-fit assignment algorithm is used. The first-fit constructs $P = \{\{\tau_1\}, \{\tau_2, \tau_3\}, \{\tau_4\}, \{\tau_5\}, \{\tau_6\}\}$ which has $K_4 = 130/70 = 1.85$, and hence, $V_4 = \lfloor 1.85 \rfloor \times 1 - 1 = 0$.*

As shown by Example 1, if the first-fit algorithm could take a look to the unassigned tasks, it could pair up τ_2 and τ_4, and hence, construct $P = \{\{\tau_1\}, \{\tau_2, \tau_4\}, \{\tau_3\}, \{\tau_5\}, \{\tau_6\}\}$. Then the task set would become schedulable. We denote EP-RM which is used with the first-fit task assignment algorithm by EP-RM-FF.

In the next step, we present EP-RM Wise-fit (EP-RM-WF) algorithm by introducing another condition to the first-fit algorithm. EP-RM-WF guarantee that every task set which satisfies Theorem 4 satisfies Theorem 7 as well. In this algorithm, task τ_i is assigned to priority group P_j if (13) holds and we have

$$\forall \tau_x \in \{\tau'_1, \tau'_2, \ldots, \tau'_m, \tau_{i+1}, \tau_{i+2}, \ldots, \tau_n\}, \text{ Theorem 4 holds} \quad (14)$$

where $\tau'_x = (C_x, T_1^x)$ (for $1 \leq x \leq m$ and $x \neq j$) are the tasks constructed from the priority groups. We have $\tau'_j = (C_j + c_i, T_1^j)$ for the priority group P_j. Condition (14) guarantees that the remaining unassigned tasks will not be unschedulable if task τ_i is added to P_j. The following theorem shows that EP-RM-WF improves P-RM's schedulability.

THEOREM 8. *EP-RM-WF improves schedulability of P-RM in Theorem 4.*

PROOF. For the proof, first we show that any task set schedulable by P-RM according to Theorem 4, is schedulable by EP-RM-WF according to Theorem 7, and then, show that there are task sets which are schedulable by EP-RM-WF and not by P-RM. If for every task in this task set, conditions of Theorem 4 are satisfied, P-RM can successfully schedule the task set. In the worst-case, if none of those tasks are grouped in any of the other priority groups, they create their own priority group (which is identified by their period and execution time). Then we will have n priority groups each with one task. Since there is no tail task, there is no need to verify (12). Besides, the schedulability condition is the same as having n tasks with exact parameters of τ, the schedulability condition in (7) and (11) becomes identical.

On the one hand, based on (13) and (14), when a task is assigned to an existing priority group, none of the future or the existing priority groups will violate the schedulability condition (11), which is consistent with (7). Thus, the assignments will not violate conditions of Theorem 7. On the other hand, if because of grouping the tasks into priority groups, some of them are removed from the chain of the tasks which affect V_i, more chances will appear for other tasks to be scheduled. For instance, the task set in Example 1 is not schedulable by P-RM or EP-RM-FF, yet it is schedulable by EP-RM-WF because it groups τ_2 and τ_4 together, which will result in $K_5 = 130/40 = 3.25$ and $V_5 = \lfloor 3.25 \rfloor \times 0.5 - 1 = 0.5$. □

It is worth mentioning that the goal of Theorem 8 is to show that the schedulability of P-RM is improved using priority groups and wise-fit task assignment algorithm. In the

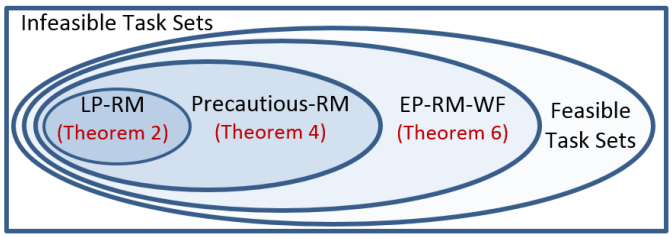

Figure 6: The relation between schedulability of the algorithms based on Theorem 2 for LP-RM, Theorem 4 for P-RM, and Theorem 6 for EP-RM-WF

next section, we discuss more about the dominance relation between the schedulability tests and the scheduling algorithms.

4.3 Properties of the Algorithms

We start this section by showing that each task set which is schedulable by LP-RM according to the schedulability test in Theorem 2, is schedulable by P-RM based on Theorem 4.

THEOREM 9. *The schedulability conditions for P-RM in Theorem 4 dominate the schedulability conditions of LP-RM in Theorem 2.*

PROOF. Assuming $c_i > T_1 - c_1$, for $1 < i \leq n$, equation (2) becomes similar to (6). Since according to (7) and (5), the schedulability of P-RM and LP-RM is based on verifying the existence of $v_i \geq 0.5$, both of them almost behave the same. If some tasks have smaller execution time than $T_1 - c_1$, they have greater v_i in P-RM. Assume the new value of vacant intervals for τ_i is shown by v'_i, thus, for each v'_j $(i < j \leq n)$ which can be obtained for the lower priority tasks, we will have $v'_j > v_j$. Consequently, the chance that we have $v'_j \geq 0.5$ increases. Finally, the following task set $\tau = \{\tau_1, \tau_2, \tau_3\}$, $\tau_1 = (10, 2)$, $\tau_2 = (5, 30)$, and $\tau_3 = (4, 30)$ is an example which shows P-RM can schedule a task set which is not schedulable by LP-RM, and consequently, it dominates LP-RM. □

From Theorem 8 and Theorem 9 we can deduce that EP-RM-WF dominates the schedulability conditions of LP-RM too. Figure 6 shows the relation between the algorithms.

In the next theorem we show that a task set with period ratios greater than or equal to 3, is schedulable by LP-RM, P-RM, and EP-RM-WF if it fulfills the necessary conditions of schedulability, i.e., (3) and (4). Note that previously [6] and [15] have proven the feasibility of such task sets only in case of harmonic tasks which have synchronous release.

THEOREM 10. *A loose-harmonic task set τ that satisfies (3), and (4) and has $k_i \geq 3$ $(\forall i, 1 < i \leq n)$, is schedulable by P-RM, LP-RM, and EP-RM-WF.*

PROOF. We show that the given task set is schedulable by LP-RM, hence, it is schedulable by the other algorithms as well. If $k_i \geq 3$, we have $\lfloor k_i \rfloor \geq 3$, thus, $v_2 \geq 3 \times 0.5 - 1 \Rightarrow v_2 \geq 0.5$, which guarantees the schedulability of τ_2. This process can be repeated for other tasks as well. Finally, task τ_n will have at least 0.5 units of vacant intervals, which guarantees the schedulability of the task set even if T_n/T_1 is not an even number. □

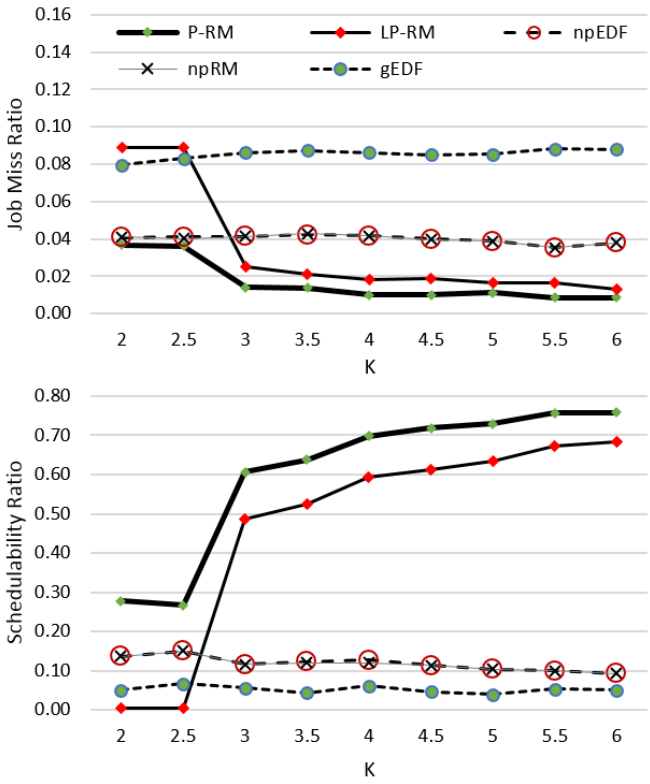

Figure 7: Results of the experiments for task sets having conditions (3) and (4) for different values of $K = \max\{k_i\}_{1<i\leq n}$, the largest period ratio.

Computational complexity of the schedulability tests for LP-RM and P-RM in Theorems 2 and 4 is $O(n)$ because v_i can be calculated in $O(1)$ for each task, and we have n tasks. For the same reason, conditions of Theorem 7 can be verified in $O(n)$ as well.

A naive implementation of the priority assignment algorithms in Sect. 4.2 can run in $O(n^3)$. We need to assign n tasks and we check if each of the tasks can fit into any of the existing priority groups (which are at most n). The schedulability is evaluated by Theorem 7 or condition (13) which can be verified in $O(n)$. Thus, the final complexity is $O(n^3)$, though it might be possible to have a more efficient implementation and avoid re-calculation of V_i and v_i values for the unchanged priority groups.

In the next section, we evaluate the performance of the algorithms and discuss their efficiency.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of P-RM and LP-RM in comparison with other existing non-preemptive scheduling algorithms npEDF, npRM, and gEDF [12]. In the first set of experiments, performance measures are job miss ratio (the ratio of missed jobs to the total number of jobs in one hyperperiod), and schedulability ratio (the ratio of successfully scheduled task sets to the total number of generated task sets). We generate random task sets with necessary conditions of schedulability, i.e., (3) and (4). Each simulation experiment averages 500 simulation runs (each including 7 randomly generated tasks). Due to the space

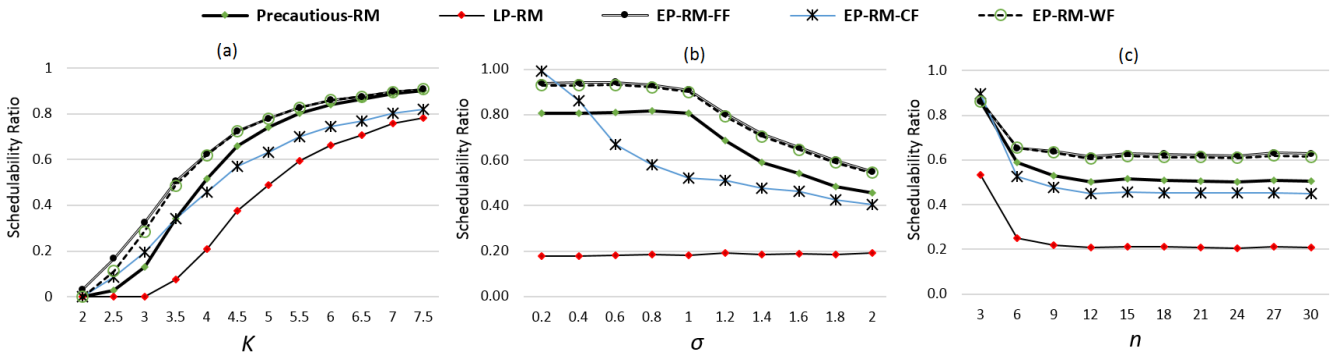

Figure 8: The effect of K, n, and σ on the performance of the schedulability tests; Theorem 2, Theorem 4, and Theorem 7

limitation, we did not report the results for larger task sets, however, the performance of the algorithms are almost the same as shown in the second set of experiments to compare schedulability tests. For confidence level 0.95, obtained results had confidence interval ± 0.05.

The parameter of the first experiment is the wideness of period ratios, i.e., $K = \max\{T_i/T_{i-1}\}_{1<i\leq n}$. In the first step, we have chosen random values for T_1 and u_1 from ranges $[1, 10]$ and $[0.01, 0.99]$ with uniform distribution, respectively, and then, we have assigned $c_1 = u_1 T_1$. Afterwards, for every task τ_i $(1 < i \leq n)$ we have selected random $k_i \in [1, K]$ with uniform distribution. Periods are constructed as $T_i = \lfloor (k_i T_{i-1})/T_1 \rfloor * T_1$, hence, they are integer multiples of T_1. The execution times have been selected randomly with the uniform distribution from $[0.01, 2(T_1 - c_1)]$. Because of the choice of periods, hyperperiod might become very large. In this experiment, we ignore the task sets with more than 10000 jobs per hyperperiod.

As shown in Figure 7, the schedulability ratio of P-RM and LP-RM considerably increases with the increase in K because there will be more vacant intervals in the hyperperiod. Unlike the work-conserving algorithms such as npEDF and npRM, P-RM and LP-RM exploit the existing vacant intervals to increase the schedulability. It is notable that before $K = 3$, LP-RM is unable to guarantee the schedulability because it always spends one vacant interval for each task. According to (2), if $k_2 < 3$, τ_2 has zero vacant intervals, and hence, the task set will not be schedulable by LP-RM. Unlike LP-RM, P-RM tries to share vacant intervals between the tasks, and hence, has better schedulability ratio among the other algorithms. It is worth mentioning that in the unschedulable task sets, miss ratio of LP-RM and P-RM is considerably small (about 2%). High miss ratio in gEDF comes from the fact that it prioritizes shorter tasks over the longer tasks, and hence, in cases where c_1 is smaller than other tasks, it misses many jobs of the other tasks, and in cases where c_1 is larger than the others, it misses many jobs of τ_1.

In the next experiment, we evaluate performance of the schedulability tests. To evaluate wise-fit in EP-RM-WF, we have also implemented two other versions of the first-fit. The first one, which is denoted by FF, applies condition (13) while the second one which is denoted by CF (or carefree-fit) only applies condition $C_i \leq 2(T_1 - c_1)$ where C_i is the size of the current priority group P_i. We study the effect of K, n, and $\sigma = C/(2(T_1 - c_1))$, where σ is the ratio of $C = \max\{c_i\}_{1<i\leq n}$ to the slack of the first task. Figure 8 shows the results of these experiments. For each data point, 10,000 random task sets with $u \leq 1$ and $u_1 \in [0.01, 0.99]$ are generated. Since the schedulability tests do not depend on the hyperperiod, in this experiment we did not apply any limitation on the value of the hyperperiod. Other setup values for Figure 8-a are $n = 10$ and $0.01 \leq c_i \leq 2(T_1 - c_1)$, for Figure 8-b are $1 \leq k_i \leq 4.0$ and $n = 10$, and for Figure 8-c are $1 \leq k_i \leq 4.0$ and $0.01 \leq c_i \leq 2(T_1 - c_1)$. In Figure 8-b, the execution times are assigned with uniform distribution from range $[0.01, \sigma \times 2(T_1 - c_1)]$.

As shown in Figure 8-a, schedulability increases with the increase in K because period ratio increases the amount of vacant intervals, and hence, rises the chance to have $v_i \geq 0.5$ even for LP-RM scheduling algorithm. For $K > 5.5$, the performance of P-RM is almost the same as EP-RM. As it can be seen, there is a small difference between EP-RM-WF and EP-RM-FF when $K < 3$. Those task sets are harder to schedule because they have fewer vacant intervals. EP-RM-FF algorithm does not need to apply condition (14), and hence, it is able to group more tasks into a priority group than wise-fit. That is why it has higher schedulability ratio when $K < 3$. As it can be seen in the figure, EP-RM-CF, which does not apply (13) has lower schedulability ratio than P-RM from $K \geq 3.5$. It shows the importance of verifying schedulability conditions for the existing priority groups during the task assignment process. We have also implemented the best-fit algorithm, however, because in almost all experiments, it was the same as EP-RM-FF, we did not report its results. One of the reasons was that there were usually small number of potential priority groups which could accept an unassigned task, and hence, the best fit was the first fit.

In Figure 8-b, the schedulability decreases with the increase in σ, especially for $\sigma > 1$. The reason is that tasks with smaller execution times can fit in the existing priority groups, hence, EP-RM produces smaller number of priority groups with larger number of tasks. As it can be seen, the difference between EP-RM-CF and the other two fitting algorithms is large when $\sigma > 0.4$. It shows the importance of applying condition (13) during the task assignment process. However, as we see for $\sigma = 0.2$, a heuristic algorithm such as CF which does not care about (13) has higher schedulability ratio than the others. It is worth reminding that

the only task assignment algorithm which can guarantee to have schedulability ratio higher than P-RM is EP-RM-WF as there are task sets which cannot be scheduled by FF.

Finally Figure 8-c shows that n does not affect the result of each individual schedulability test, especially when it is greater than 6. As we have expected, task grouping improves the schedulability if it guarantees condition (13). The difference between EP-RM-WF and P-RM is about 16% in average and it shows that task grouping can efficiently increase the schedulability. However, as it can be seen in the difference between P-RM and EP-RM-CF, if task grouping is not done properly, it may even reduce the schedulability.

6. SUMMARY AND CONCLUSION

In this paper, we have considered the problem of non-preemptive scheduling in uniprocessor systems with periodic tasks and implicit deadlines with periods integer multiples of the smallest period. We call this task set loose-harmonic. We have focused on non-work-conserving scheduling algorithms, i.e., the algorithms that may insert idle intervals in the schedule even though there might be pending jobs in the system.

We have extended the schedulability of P-RM and its simpler version LP-RM from harmonic tasks to loose-harmonic tasks. The new schedulability tests have $O(n)$ computational complexity. Moreover, to extend the scope of schedulable task sets we have presented a new scheduling algorithm called EP-RM. It allows groups of tasks to share the same priority, then forces them to be scheduled only if the task with the smallest period in the group is scheduled. By introducing a task assignment algorithm called wise-fit, we have theoretically proven the dominance of our schedulability test for EP-RM over P-RM. Furthermore, we have shown that if the ratio of two consecutive periods is greater than or equal to 3, LP-RM, P-RM, and EP-RM (with wise-fit priority assignment) are able to schedule any feasible loose-harmonic task set. Our experimental results have shown that P-RM has in average, about 40% higher schedulability ratio than other work-conserving algorithms such as npRM and npEDF for the task sets which satisfy necessary schedulability conditions.

7. REFERENCES

[1] A. Al-Sheikh, O. Brun, P. E. Hladik, and B. J. Prabhu. A best-response algorithm for multiprocessor periodic scheduling. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 228–237, 2011.

[2] B. Andersson and E. Tovar. The utilization bound of non-preemptive rate-monotonic scheduling in Controller Area Networks is 25%. In *IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 11–18, 2009.

[3] S. Anssi, S. Kuntz, S. Gérard, and F. Terrier. On the gap between schedulability tests and an automotive task model. *Journal of Systems Architecture*, 59(6):341–350, 2013.

[4] E. Bini and A. Cervin. Delay-aware period assignment in control systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 291–300, 2008.

[5] G. C. Buttazzo. Rate monotonic vs. edf: Judgment day. *Lecture Notes in Computer Science*, 2855:67–83, 2003.

[6] Y. Cai and M. C. Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica*, 15(6):572–599, 1996.

[7] J. S. Deogun and M. C. Kong. On periodic scheduling of time-critical tasks. In *IFIP World Computer Congress*, pages 791–796, 1986.

[8] C. Ekelin. Clairvoyant non-preemptive EDF scheduling. *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 23–32, 2006.

[9] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uni-processor scheduling. Technical report, INRIA Research Report nRR2966, 1996.

[10] M. Grenier and N. Navet. Fine-tuning MAC-level protocols for optimized real-time QoS. *IEEE Transaction on Industrial Informatics*, 4(1):6–15, 2008.

[11] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 129–139, 1991.

[12] W. Li, K. Kavi, and R. Akl. A non-preemptive scheduling algorithm for soft real-time systems. *Computers and Electrical Engineering*, 33(1):12–29, 2007.

[13] M. Marouf and Y. Sorel. Schedulability conditions for non-preemptive hard real-time tasks with strict period schedulability analysis. In *Conference on Real-Time and Network Systems (RTNS)*, pages 50–58, 2010.

[14] M. Nasri, S. Baruah, G. Fohler, and M. Kargahi. On the Optimality of RM and EDF for Non-Preemptive Real-Time Harmonic Tasks. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 211–220. ACM, 2014.

[15] M. Nasri and M. Kargahi. Precautious-RM: a predictable non-preemptive scheduling algorithm for harmonic tasks. *Real-Time Systems*, 50(4):548–584, 2014.

[16] J. R. Nawrocki, A. Czajka, and W. Complak. Scheduling cyclic tasks with binary periods. *Information Processing Letters*, 65(4):173–178, 1998.

[17] M. Park. Non-preemptive fixed priority scheduling of hard real-time periodic tasks. In *International Conference on Computational Science*, pages 881–888, 2007.

[18] H. Ramaprasad and F. Mueller. Bounding worst-case response time for tasks with non-preemptive regions. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, page 58–67, 2008.

[19] Z. Sahraoui, E. Grolleau, M. A. Nacer, D. Mehdi, and H. Bauer. Antinomy Between Schedulability and Quality of Control Using a Feedback Scheduler. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 171–180. ACM, 2014.

[20] K. M. Zuberi and K. G. Shin. Non-preemptive Scheduling of Messages on Controller Area Network for Real-time Control Applications. In *IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 240–249. IEEE Computer Society, 1995.