Real-Time Syst (2014) 50:548–584
DOI 10.1007/s11241-014-9203-y

Precautious-RM: a predictable non-preemptive scheduling algorithm for harmonic tasks

Mitra Nasri · Mehdi Kargahi

Published online: 3 June 2014

Abstract A major requirement of many real-time embedded systems is to have time-predictable interaction with the environment. More specifically, they need fixed or small sampling and I/O delays, and they cannot cope with large delay jitters. Non-preemptive execution is a known method to reduce the latter delay; however, the corresponding scheduling problem is NP-Hard for periodic tasks. In this paper, we present Precautious-RM as a predictable linear-time online non-preemptive scheduling algorithm for harmonic tasks which can also deal with the former delay, namely sampling delay. We derive conditions of optimality of Precautious-RM and show that satisfying those conditions, tight bounds for best- and worst-case response times of the tasks can be calculated in polynomial-time. More importantly, response time jitter of the tasks is analyzed and it is proven that under specific conditions, each task has either one or two values for response time, which leads to improving the predictability of the system interaction with the environment. Simulation results demonstrate efficiency of Precautious-RM in increasing accuracy of control applications.

Keywords Non-preemptive scheduling · Harmonic tasks · Delay and jitter · Accuracy enhancement · Control applications · Real-time systems

M. Nasri · M. Kargahi (✉)
School of Electrical and Computer Engineering, College of Engineering,
University of Tehran, Tehran, Iran
e-mail: kargahi@ut.ac.ir

M. Nasri
Technical University of Kaiserslautern, Kaiserslautern, Germany
e-mail: mitra.nasri@ut.ac.ir

M. Kargahi
School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

1 Introduction

Real-time embedded systems generally need predictable interaction with the environment. Delay and jitter of the instants of the interactions in many such systems (e.g., control systems, avionics, and robotics) adversely affect the system quality of service (Buttazzo and Cervin 2007; Bini and Cervin 2008) and accuracy (Nilsson et al. 2010; Nasri et al. 2012; Nasri and Kargahi 2012). More specifically, these systems need either small or fixed values of (i) sampling delay (the gap between release-time and sampling (start-time) of a task) and (ii) I/O delay (the delay between the task sampling and actuation). It is worth mentioning that, if the values of sampling and I/O delays are either constant or known at design-time, it is possible in many real-time embedded systems to tolerate these delays in the application-level (Marti et al. 2001, 2002; Nilsson et al. 2010).

One approach to reduce I/O delay is what used in limited preemption (Wu and Bertogna 2009) which tries to control preemptions. However, due to the preemptions which might occur during the execution of different instances of a task, such an approach cannot eradicate or minimize the I/O delay jitter. One known method to avoid these extra I/O delay and/or jitter is non-preemptive execution (Buttazzo et al. 2013). This type of execution in avionics, robotics, and cell phone applications (Piaggio et al. 2000; Park 2007; Al-Sheikh et al. 2011) have had benefits including shorter job execution, more efficient use of system cache and pipeline (Bastoni 2010), and more precise estimation of the job (worst-case) execution time (Buttazzo et al. 2013), leading to more predictable I/O delay.

Previously, it has been proven that non-preemptive scheduling of periodic tasks is NP-Hard (Jeffay et al. 1991). However, there have been studies investigating sufficient conditions for schedulability of periodic tasks according to non-preemptive rate-monotonic (npRM) (Park 2007; Andersson and Tovar 2009), non-preemptive fixed priority scheduling algorithms (Davis et al. 2007), and non-preemptive earliest-deadline-first (npEDF) (Jeffay et al. 1991; Georges et al. 2000). Also, schedulability analysis of non-preemptive tasks with strict periods (i.e., when the start-time of all jobs is strictly periodic) have been studied by Marouf and Sorel (2010). In spite of the hardness of non-preemptive scheduling, there have been attempts to find efficient heuristics for the problem; clairvoyance EDF (cEDF) (Ekelin 2006) and group-based EDF (gEDF) (Li et al. 2007) are major examples. cEDF, which has been proposed for firm real-time tasks (Ekelin 2006), uses a form of look ahead to determine the appropriate time to insert an idle interval before starting the execution of an urgent task. gEDF, which has been presented for soft real-time systems, first groups the tasks based on the closeness of their deadlines, and then selects the task with the smallest execution time among the tasks of the group with the earliest deadline.

Harmonic tasks, namely, the set of tasks that each period is an integer divisor of larger periods, are known as an appropriate model for large spectrum of applications such as avionics, submarines, and robotics (Busquets-Mataix et al. 1996; Bate et al. 1996; Li et al. 2003; Anssi et al. 2013) as well as control systems with nested feedback loops (Fu et al. 2010). Accordingly, there have been efforts to convert periodic tasks into harmonic ones (Kuo and Mok 1991; Han and Tyan 1997), which have also been employed in dwell tasks of Radar systems (Shih et al. 2003). However, even when the

periods are harmonic, non-preemptive scheduling of the tasks is NP-Hard (Cai and Kong 1996).

One pioneering algorithm to optimally schedule non-preemptive harmonic tasks has been presented in Deogun and Kong (1986), where each task period is K times of the immediate task with the smaller period ($K \in \{3, 4, \ldots\}$ is a constant integer). Later, Cai and Kong (1996) have shown that this problem is NP-hard in the general case, i.e., when the period ratios are arbitrary integer values greater than or equal to 1. In the same work, they have also introduced an optimal scheduling algorithm for binary harmonic tasks where each period is assigned to a unique task and is two times of the immediate smaller period. Time-complexity of this algorithm is $O(2^n)$, where n is the number of tasks. As an extension of the work, they merged the algorithm of Deogun and Kong (1986) with their own algorithm for offline scheduling of a particular case of harmonic tasks where the first m tasks with smaller periods have period ratios 2 and the other $n - m$ tasks have period ratios greater than or equal to 3. Afterwards, Nawrocki et al. (1998) have proven that when the ratio between periods is a power of 2, still the problem is NP-hard.

Non-preemptive execution, as mentioned above, can be used to have more predictable execution time (and thus, I/O delay). However, to increase the system predictability, it is also required to limit the sampling delay and jitter as well (Marti et al. 2001). As mentioned in the following, this study pays attention to both these concerns together.

In this paper, we present Precautious-RM, an online non-preemptive scheduling algorithm with linear-time computational complexity, to efficiently schedule harmonic real-time tasks. We sketch theoretical conditions for optimality (i.e., to have a schedule with no deadline miss) of Precautious-RM. We prove that when those conditions hold, it is possible to derive tight bounds for the worst-case response time (WCRT) of the tasks in polynomial-time. It is worth mentioning that calculating tight bounds of WCRT can be an intractable problem, even for preemptive scheduling algorithms such as RM or EDF (Eisenbrand and Rothvoß 2008, 2010). Further, we present sufficient conditions to have no response time jitter (RTJ) for a task. According to Buttazzo and Cervin (2007), RTJ which is the difference between the task WCRT and best-case response time (BCRT), affects the performance of many control applications.

As a major difference to the previous works, Precautious-RM is able to increase the predictability of the real-time systems by reducing the diversity of possible sampling delays, while at the same time it eliminates extra I/O delay and jitter through non-preemptive execution and it guarantees all deadlines. Efficiency of this algorithm has been evaluated by simulation experiments with various feasible and infeasible harmonic tasks through comparing miss ratio (MR), response time, RTJ, and response time diversity. In addition, a benchmark control application has been simulated to show the efficiency of the solution in control systems. The results confirm improvements in the quality of control (QoC), RTJ, and MR comparing to some known non-preemptive scheduling algorithms in the literature.

The remainder of this paper is structured as follows: Sect. 2 introduces the system model. Precautious-RM is presented in Sect. 3. Then, in Sect. 4, we provide the optimality conditions of the algorithm. A polynomial-time algorithm to derive tight bounds of the BCRT and WCRT is then presented in Sect. 5. Sect. 6 summarizes the

theoretical achievements of the paper. In Sect. 7, Precautious-RM has been evaluated through extensive simulations using a benchmark control application. Finally, the paper is concluded in Sect. 8.

2 System model

We consider a single-processor real-time embedded system running a set of harmonic independent non-preemptive real-time tasks $T = \{T_1, T_2, \ldots, T_n\}$. Each task $T_i (1 \leq i \leq n)$ is identified by (p_i, c_i) where $p_i \in \mathbb{R}^+$ is the period and $c_i \in \mathbb{R}^+$ is the execution time. We assume that deadlines are implicit, thus the period of each task is considered as its relative deadline. This assumption is needed by many of the control applications because their stability relies on the guaranteed periodic behavior (Eker et al. 2000).

We assume all of the tasks are in-phase, i.e., they start synchronously at time 0, and for each task T_i, $1 \leq i < n$, $p_i = k_i p_{i+1}$ where k_i is an arbitrary positive integer value. p_n is the smallest period and p_1 is the hyperperiod (the least common multiplier of the task periods). It is worth mentioning that in an in-phase set of harmonic tasks, each task is released along with the other tasks with smaller periods. We suppose that the tasks perform their input upon start of their non-preemptive execution and do their output upon completion. Also, it is considered that the I/O devices are capable to perform immediate I/O operations (the time needed to perform those operations have been ignored).

3 Precautious-RM algorithm

In this section, Precautious-RM is presented as a non-work-conserving online scheduling algorithm with linear time complexity. The main difference between Precautious-RM and npRM is that the former uses idle intervals to guarantee that the deadline of the incoming instance of T_n will be met. This algorithm is activated whenever a task instance finishes or a scheduled idle interval ends. Algorithm 1 presents pseudo-code of Precautious-RM.

Suppose that the algorithm is activated at time t. It searches according to the rate monotonic priorities for a candidate task which has been released before t but has not been executed yet (Lines 2 and 3 of Algorithm 1). Since relative deadlines are not longer than periods, at any time t, each task has exactly one released instance which may or may not be executed before t. Line 3 verifies that the current instance of the candidate task has not been executed before. This line can be implemented by checking a simple list with binary values which keeps track of the executed and non-executed tasks in the system.

Then, the candidate task is scheduled if it can be executed non-preemptively while the next instance of T_n is able to meet its deadline. Assume that the next instance of T_n will be released at r_n^{next}. First we evaluate $t + c_i \leq r_n^{next}$ where c_i is the execution time of task T_i, i.e., the highest priority task in the system. If the inequality holds, it means that T_i will be finished before the next release of T_n; thus, T_i can be scheduled.

Algorithm 1 Precautious-RM

Input T, t
T is the task set, and $t \in R$ is the current time
Output S, t_{next}
S is the next task to be scheduled ($S \leftarrow null$ means that processor remains idle), and
$t_{next} \in R$ is the next time to activate the scheduler.

```
1: r_n^next ← next release time of T_n according to t
2: for i ← n downto 1 do
3:     if (current instance of T_i has not been executed) then
4:         if ((t + c_i ≤ r_n^next) or (the latest executed task is T_n and t + c_i ≤ r_n^next + p_n − c_n)) then
5:             S ← T_i
6:             t_next ← t + c_i
7:             return S, t_next
8:         else
9:             break
10:        end if
11:    end if
12: end for
13: S ← null
14: t_next ← r_n^next
15: return S, t_next
```

Second, we go further and calculate $r_n^{next} + p_n - c_n$ as the latest possible time to start the next instance of T_n. If $t + c_i \leq r_n^{next} + p_n - c_n$ holds, the next T_n is still able to meet its deadline, however, we need an additional condition to control the number of low priority tasks which will be scheduled before the next instance of T_n. For example, when the task set is binary, i.e., $k_j = 2, 1 \leq j < n$, scheduling more than one long low priority task between two instances of T_n will cause deadline miss for further high priority tasks with shorter periods. This case will be discussed with more details in Sect. 4.2, where optimality of Precautious-RM is studied for binary task sets.

It is worth mentioning that Precautious-RM is not greedy about scheduling possible low priority jobs. In fact, if there is a high priority job in the system which cannot be scheduled in the current situation (due to Line 4 of Algorithm 1), Precautious-RM stops the search and just inserts an idle interval (in Lines 13 and 14 of Algorithm 1). This decision takes place instead of continuing the search and finding a lower priority job with an appropriate length to fit in this interval. In other words, rather than following a greedy manner, Precautious-RM warily schedules a task when it looks up for a candidate task, since it considers possible deadline misses in the future.

Computational complexity of Precautious-RM is $O(n)$, where n is the number of tasks in the system. This can be verified by the loop in Lines 2 to 12 of Algorithm 1. It is worth mentioning that Precautious-RM inserts idle times and will be activated when these idle times are ended, however, since each idle interval ends with the next release of T_n, the number of inserted idle intervals during one hyperperiod is at most the same as the number of instances of task T_n, namely p_1/p_n which results to reasonable number of activations of the algorithm during one hyperperiod.

4 Optimality of Precautious-RM

In this section, we sketch three task settings where Precautious-RM can perform optimally, i.e., it presents a feasible schedule if such a schedule exists: (i) harmonic tasks with arbitrary period ratios greater than 2, (ii) binary harmonic tasks with constant period ratio 2, and (iii) general harmonic tasks satisfying some sufficient constraints with arbitrary period ratios greater than or equal to 1.

4.1 Harmonic tasks with arbitrary period ratios greater than two

The following theorem describes the optimality conditions of Precautious-RM for harmonic tasks with period ratios >2. The same theorem gives also a (not necessarily tight) bound on WCRT of the tasks. Later in Sect. 5, the tight bound of WCRT will be derived.

Theorem 1 *Precautious-RM can feasibly schedule a non-preemptive task set T, if the following conditions hold:*

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \tag{1}$$

$$c_i \leq 2(p_n - c_n), \quad 1 \leq i < n \tag{2}$$

$$k_i > 2, \quad 1 \leq i < n \tag{3}$$

Then, the worst-case response time of task T_i is obtained as:

$$WCRT_i = \begin{cases} p_n, & i = n \\ 2p_{i+1} - p_n + c_n + c_i, & 1 < i < n \\ p_2 - p_n + c_n + c_1, & i = 1 \end{cases} \tag{4}$$

Proof To prove that Precautious-RM maintains feasibility of the schedule, it is enough to show that WCRT of each task is smaller than its relative deadline. In the first step we show that WCRT of equation (4) is always smaller than or equal to the relative deadlines of the tasks, then in the second step we prove that Precautious-RM is able to guarantee those WCRTs. In equation (4), the WCRT of T_n is equal to its deadline, and for the other tasks it is $2p_{i+1} - p_n + c_n + c_i$. We replace c_i by its upper bound in (2), thus we have $2p_{i+1} - p_n + c_n + c_i \leq 2p_{i+1} + p_n - c_n \leq 2p_{i+1} + p_n$. Also $p_n \leq p_{i+1}$ thus $2p_{i+1} + p_n \leq 3p_{i+1}$ which is smaller than or equal to the deadline of the tasks because of Condition (3). Consequently, equation (4) is always smaller than or equal to the task relative deadlines.

In the second step, using a strong induction we prove that response time of each task is bounded by Eq. (4).

Base of induction For T_n and T_{n-1}, if the processor is idle at the release of T_{n-1}, the response time of T_{n-1} will be $c_n + c_{n-1}$ and there will be an idle interval with length $p_n - c_n$ before the third release of T_n. Otherwise, response time of T_{n-1} will be bounded by $p_n + c_n + c_{n-1}$. Note that both response time values are smaller than or

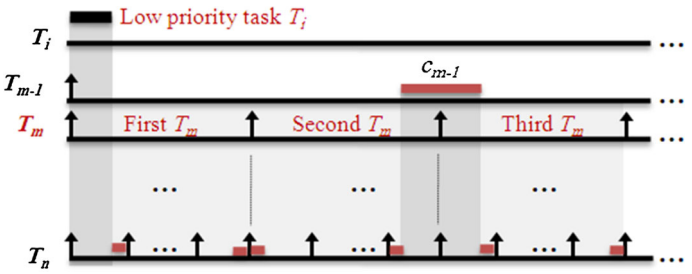

Fig. 1 Schedule of T_{m-1} when the processor is busy at its release

equal to what calculated in (4). Lemma 4 in the Appendix proves the aforementioned statements for the base of this induction.

Induction hypothesis We assume that Precautious-RM can successfully schedule every task set from T_n to T_m, $1 < m < n$ such that (4) holds. Also, we suppose that if the processor is busy at the start of each of the tasks which are released simultaneously with the release of T_i, $m \leq i < n$, their response time is at most $2p_{i+1} - p_n + c_n + c_i$. Otherwise, their response time is smaller than $p_{i+1} - p_n + c_n + c_i$ and these tasks will be finished $p_n - c_n$ units of time before the third release of T_{i+1}.

Inductive step Task T_{m-1} can be successfully scheduled. Also if the processor is busy at the release of T_{m-1}, its response time is at most $2p_m - p_n + c_n + c_{m-1}$. Otherwise, its response time is smaller than $p_m - p_n + c_n + c_{m-1}$ and there will be an empty interval with length $p_n - c_n$ before the third release of T_m during the release of T_{m-1}.

At first, we assume that the processor is busy when T_{m-1} is released (see Fig. 1). According to the induction hypothesis, the first T_m and all of the tasks with periods smaller than T_m can be successfully scheduled with the promised response times even if the processor is busy for at most $p_n - c_n$ units of time. For the worst-case, we assume that there is no chance for T_{m-1} to be scheduled before the deadline of the first release of T_m. However, at the beginning of the second release of T_m, the processor will be idle because of Line 8 of Algorithm 1. Thus, as said by the hypothesis of the induction, just before the third release of T_m, there will be an idle interval with length $p_n - c_n$. Also for the same reason, it is guaranteed that all other instances of the tasks T_j, $m \leq j \leq n$ can be successfully scheduled before that time; therefore, at that time, T_{m-1} has the highest priority in the system and its execution time is smaller than $2(p_n - c_n)$. It means that T_{m-1} will be selected by Line 4 of Algorithm 1. In this case, response time of this task is $2p_m - p_n + c_n + c_{m-1}$. When the execution of T_{m-1} is finished, all tasks which are released synchronously with the next release of T_m can be scheduled successfully because according to the hypothesis of the induction, these tasks can be successfully scheduled with the promised response time even if, at their release, the processor is blocked by the execution of a low priority task.

Now we consider the case that at the release of T_{m-1}, the processor is idle (as shown in Fig. 2). We assume $\sigma = p_m$ is the time of the second release of T_m (during the current release of T_{m-1}). According to the induction hypothesis, at time $\sigma - (p_n - c_n)$ there will be an idle interval with length $p_n - c_n$. Thus, if T_{m-1} could not find a chance to be scheduled by Precautious-RM before that time, the algorithm schedules

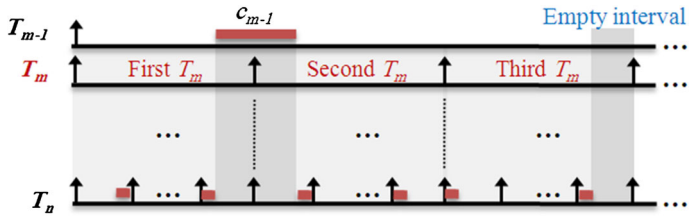

Fig. 2 Schedule of T_{m-1} when the processor is idle at its release

it at $\sigma - (p_n - c_n)$. The reason is that, at this time, task T_{m-1} is the highest priority task which is not executed yet. Also we have $c_{m-1} \leq 2(p_n - c_n)$, meaning that T_{m-1} will be finished before $\sigma + (p_n - c_n)$; hence, the next instance of T_n will not miss its deadline. Thus, Line 4 of Algorithm 1 is satisfied. In this case, response time of the task will be bounded by $p_m - p_n + c_n + c_{m-1}$. Moreover, since T_{m-1} has been executed before the third release of T_m, based on the induction hypothesis, there will be one idle interval with length $p_n - c_n$ just before deadline of the third T_m. This idle interval is what promised as the final requirement of the induction.

Above, we assumed that the worst-case happens for task T_{m-1} and it will be placed at the expected idle interval, just before the second or third release of task T_m. Now, we see that if the algorithm schedules task T_{m-1} earlier than that time, no other task misses its deadline due to the execution of T_{m-1}. Note that T_{m-1} is selected by the algorithm only if there is no high priority task in the ready queue of the system. We consider two cases based on the number of tasks which are released during the execution of T_{m-1}. In the first case we assume no task except T_n is released. Since in all cases, Line 4 of the Algorithm 1 guarantees deadline of the next T_n, T_{m-1} will not cause any deadline miss for the other task. In the second case we assume a set of tasks from T_n to T_i, $n < i \leq m$, are released synchronously during the execution of T_{m-1}. In this case, these tasks will be blocked by the execution of the low priority task T_{m-1}. However, in the induction hypothesis we considered that all of the tasks from T_n to T_m can be scheduled successfully even when the processor is busy at their releases. Hence, the execution of task T_{m-1} will not cause a miss for tasks T_n to T_i. In this case, response time of T_{m-1} is smaller than the expected WCRT in (4). □

It is worth mentioning that Condition (1) is necessary for schedulability of preemptive and non-preemptive systems as shown by Liu and Layland (1973) and Jeffay et al. (1991), respectively. According to Cai and Kong (1996), Condition (2) is necessary for schedulability of non-preemptive harmonic tasks. Consequently, it is possible to deduce that Precautious-RM is optimal as in fact it is able to feasibly schedule every arbitrary feasible non-preemptive harmonic task set with period ratios >2.

4.2 Optimality of Precautious-RM for binary harmonic tasks

Necessary conditions for schedulability of binary harmonic tasks ($k_i = 2$ for $1 \leq i \leq n$) have been provided by Cai and Kong (1996). They shown that the feasibility

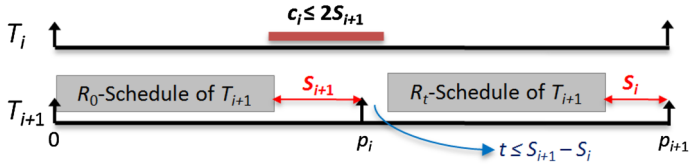

Fig. 3 Schedule of T_i for binary harmonic tasks, according to Cai and Kong (1996)

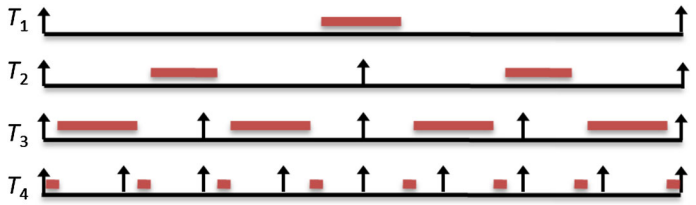

Fig. 4 An example for the algorithm propsed in Cai and Kong (1996)

condition of binary task set T is $c_i \leq 2S_{i+1}, 1 \leq i < n$, where S_i is the task slack-time defined as ($S_n = p_n - c_n$ and $I_n = 0$):

$$S_i = \begin{cases} S_{i+1}, & c_i \leq I_{i+1} + S_{i+1} \\ S_{i+1} - (c_i - (I_{i+1} + S_{i+1})), & \text{otherwise} \end{cases} \tag{5}$$

$$I_i = \begin{cases} 2I_{i+1} + (S_{i+1} - c_i), & c_i \leq I_{i+1} + S_{i+1} \\ I_{i+1}, & \text{otherwise} \end{cases} \tag{6}$$

Figure 3 shows the schedule produced for task T_i, $1 \leq i < n$. As shown in the figure, S_i is the amount of slack-time available at the end of schedule of task T_i. This slack can be used to schedule task T_{i-1}. In this figure, R_0-Schedule is a feasible schedule for all high priority tasks from T_n to T_{i+1} starting from time 0. Also R_t-schedule is a schedule for those tasks when the processor is blocked for $t = S_i - S_{i-1}$ units of time after the release of T_{i+1}. To calculate S_i at each step, a second variable is used in (5), which is called I_i and it is the total processor idle time in R_0-Schedule. Cai and Kong (1996) have proven that if for task T_i, $1 \leq i < n$, $c_i \leq 2S_{i+1}$, a feasible R_t-schedule exists for that task.

The algorithm presented by Cai and Kong (1996) schedules task T_i, $1 \leq i < n$ after finishing R_0-Schedule of T_{i+1} (right before the second release of T_{i+1}). Then an R_t-Schedule of T_{i+1} is created after the execution of T_i. Figure 4 shows an example of this schedule for a task set with 4 tasks. As shown in this figure, there is just one instance of low priority task T_i between two consecutive instances of T_n. However, if $c_i < S_{i+1}$, the algorithm wastes available interval with length $S_{i+1} - c_i$; this interval of time cannot be used to schedule other low priority tasks.

Moreover, even though Cai and Kong (1996) have shown that their algorithm is linear time in the number of task instances in one hyperperiod, it still has computational-complexity $O(2^n)$ and creates a significantly large time-table. The reason is that the number of task instances in one hyperperiod is $2^n - 1$ for binary task sets. As a result,

this method may not be used easily in systems with large number of tasks or limited memory.

Lemma 1 *For feasible task set T with n binary harmonic tasks, the schedule produced by Precautious-RM and the one created by Cai and Kong (1996) are the same if $S_{i+1} < c_i \leq 2S_{i+1}$, $1 \leq i < n$.*

Proof Since $S_{i+1} < c_i$, we have $I_i = I_{i+1} = \cdots = I_{n-1} = 0$ for every step, hence, $S_i = 2S_{i+1} - c_i$. Suppose $\sigma = p_{i+1}$ is the release time of the second instance of T_{i+1} during one release of T_i. To prove the claim, we show that Precautious-RM schedules each task T_i, $1 \leq i < n$, at time $\sigma - S_{i+1}$. Then the whole schedule will be similar to the one presented by Cai and Kong (1996) (shown in Figs. 3, 4).

According to Cai and Kong (1996), if $I_i = 0$, there is no idle time during R_0-schedule of task T_i (for more details see Lemma 3.5 of Cai and Kong (1996). It means that Precautious-RM will not schedule T_i before time $\sigma - S_{i+1}$ since it is busy with the schedule of higher priority tasks. Regarding the assumption $c_i \leq 2S_{i+1}$ of this lemma, we will have $\sigma - S_{i+1} + c_i \leq \sigma + S_{i+1}$. Since according to Eq. (5), $S_{i+1} \leq p_n - c_n$, deadline of the next task T_n releasing at time σ will not be missed by the execution of T_i. In this regards, task T_i will be scheduled by Precautious-RM since both conditions of Line 4 of Algorithm 1 are satisfied.

On the other hand, since releases of all higher priority tasks T_j, $i+1 \leq j \leq n$ are synchronous with the release of task T_{i+1} at time σ, according to Precautious-RM, there will be no chance for lower priority tasks to be scheduled after the execution of T_i. Then, according to Cai and Kong (1996), the earliest finish-time for those high priority tasks can be $p_i - S_i$ (the end of period of T_i). Consequently, the next low priority task T_{i-1} can only be scheduled at $p_i - S_i$, and this procedure is repeated by Precautious-RM. It means that the schedule produced by Precautious-RM is the same as the one produced by Cai and Kong (1996). □

Lemma 1 is the basis to prove the optimality of Precautious-RM for binary tasks. This proof is completed by the following theorem.

Theorem 2 *Precautious-RM is optimal for scheduling every binary task set T with $c_n < p_n$, $k_i = 2$, and $c_i \leq 2S_{i+1}$, $1 \leq i < n$, where S_i is calculated according to (5).*

Proof According to Lemma 1, if $c_i > S_{i+1}$, the outputs of Precautious-RM are the same as outputs of the optimal algorithm of Cai and Kong (1996), and hence, Precautious-RM is optimal. Now we show that, even in the general case where c_i can be smaller than S_{i+1}, Precautious-RM schedules T_i either sooner than or at the same time with the algorithm presented by Cai and Kong (1996). Also we show that no higher priority task misses its deadline when T_i is scheduled earlier than expected.

Suppose that Precautious-RM is activated at time t and T_i, $1 \leq i < n$ with $c_i < S_{i+1}$ is the highest priority task which has not been scheduled before t. Since $t + c_i < t + S_{i+1} < t + p_n - c_n$, the next T_n meets its deadline, thus T_i will be scheduled by Precautious-RM. Assume r_x^{next} is the time of next release of T_x, $n < x < i$ along with current release of task T_i. Two scenarios may happen; $t + c_i \leq r_x^{next}$ or $t + c_i > r_x^{next}$. In the former scenario, because of the cautiousness of Precautious-RM in Line 4 of

Algorithm 1, lower priority tasks will not be scheduled in the intervals between $t + c_i$ and r_x^{next} unless they can be finished before r_x^{next}. If they cannot be finished before then, the algorithm simply inserts an idle interval and does not schedule any task. Consequently, the processor will be free at the next release of tasks $T_j, n \leq j \leq x$ and hence, all high priority tasks can meet their deadlines (using their R_0-Schedule). In the latter scenario, since by definition we have $S_i \leq S_x$, the blocking caused by T_i is less than the blocking that high priority tasks $T_j, n \leq j \leq x$ can tolerate, and hence, they can be successfully scheduled (using their R_t-Schedule). □

4.3 General harmonic tasks and sufficient conditions for optimality of Precautious-RM

With the definition of vacant intervals in this section, we present sufficient conditions for optimal scheduling of harmonic tasks with arbitrary period ratios.

Definition 1 A vacant interval is a time interval with length $2(p_n - c_n)$ constructed between two instances of task T_n when one of them is executed at its release, and the other is possibly scheduled at the end of its period. This interval can be potentially used for scheduling task $T_i, 1 \leq i < n$ in the system.

The number of vacant intervals of task $T_i, 1 \leq i < n$ is denoted by $V_i \in \mathbb{R}^+$ and it is defined as the number of potential vacant intervals that have not been occupied by tasks with priority higher than T_i along with one release of T_i.

Lemma 2 *Given harmonic task set T with $k_i \geq 1$ and $(p_n - c_n) \leq c_i \leq 2(p_n - c_n), 1 \leq i < n$, Precautious-RM will not schedule more than one task between two consecutive executions of T_n.*

Proof Assume that the latest T_n has been released at time σ. If the processor has been busy at that time, then completion-time of T_n will be $\sigma + \epsilon + c_n$, where $\epsilon \geq 0$ is the remaining execution time of the task occupying the processor. Otherwise, T_n will be finished at $\sigma + c_n$. Assume that T_i is the current highest priority task in the system and can be selected by Precautious-RM to be scheduled after the executed T_n. Since we have $(p_n - c_n) \leq c_i$, completion-time of T_i will not be earlier than $\sigma + \epsilon + c_n + c_i$ which is after $\sigma + p_n$. Since T_n will be the task with the highest priority at that time, it will be scheduled by Precautious-RM, and as a result, just one task has been executed between two consecutive executions of T_n. □

One of the results of Lemma 2 is that each task $T_i, 1 \leq i < n$ occupies its own interval with length $2(p_n - c_n)$ which is not shared with other tasks. The following lemma counts the number of vacant intervals for the tasks.

Lemma 3 *Given a feasible harmonic task set T, with $k_i \geq 1$ and $(p_n - c_n) \leq c_i \leq 2(p_n - c_n), 1 \leq i < n$ scheduled by Precautious-RM, V_i is obtained as:*

$$V_i = \begin{cases} k_i V_{i+1} - 1, & 1 \leq i < n \\ 0.5, & i = n \end{cases} \tag{7}$$

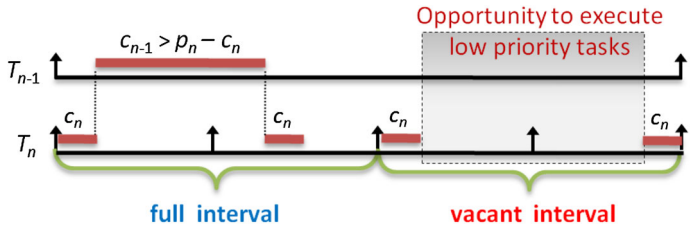

Fig. 5 Vacant intervals for task T_{n-1}

Proof Since according to Lemma 2, at most one lower priority task can be scheduled between two consecutive executions of T_n, each task T_i, $1 \leq i < n-1$ occupies one of the vacant intervals of task T_{i+1}. Now to prove (7), we use the intuitive idea of counting the number of vacant intervals under one release of each task. It is trivial to see that for task T_{n-1}, there is at most $0.5 \times k_{n-1}$ vacant intervals. Since the task set is feasible, one of those intervals has to be used for scheduling of task T_{n-1} itself (see Fig. 5). For other tasks T_i, $1 \leq i < n-1$, the vacant intervals are equal to V_{i+1} multiplied by the number of releases of that task under one release of T_i, namely, k_i. Also, since the task set is feasible, one of the vacant intervals has to be spent for scheduling task T_i itself (otherwise it will miss its deadline); thus, we have $V_i = k_i V_{i+1} - 1$. □

Now, using Lemma 2 and 3, we depict a more general case for optimality of Precautious-RM.

Theorem 3 *Given non-preemptive harmonic task set T with $k_i \geq 1, 1 \leq i < n$, Precautious-RM performs optimally if the following conditions hold:*

$$\sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \tag{8}$$

$$\begin{cases} V_i > 0, & 1 < i < n \\ V_i \geq 0, & i = 1 \end{cases} \tag{9}$$

$$(p_n - c_n) \leq c_i \leq 2(p_n - c_n) \tag{10}$$

Proof As shown by Liu and Layland (1973), Condition (8) is necessary for schedulability of the task set. Also according to Theorem 1, the right hand side of (10), namely, $c_i \leq 2(p_n - c_n), 1 \leq i < n$ is a must for schedulability of the harmonic tasks. On the other hand, (9) confirms that there would be enough time gap to schedule $T_i, 1 \leq i < n-1$; because in the definition of the number of vacant intervals of T_i, one time slot has been reserved for the task itself. That slot is the first vacant interval of task T_{i+1} and it is used by Precautious-RM to schedule task T_i. Consequently, all tasks can be scheduled before their deadlines as long as their $V_i > 0$. The exception is T_1 which can have $V_1 = 0$, meaning that no vacant interval has been left at the end of p_1. □

Since according to Lemma 2 tasks can only be executed in vacant intervals, when (10) holds, (9) is a necessary schedulability condition for Precautious-RM.

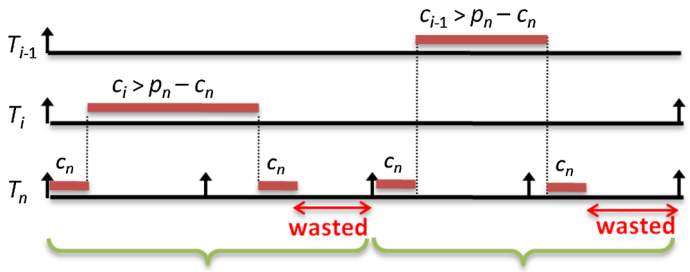

Fig. 6 An example of wasted utilization within a vacant interval

Despite its relative generality, Theorem 3 cannot deal with $k_{n-1} = 1$, because in that case we will have $V_{n-1} = 0.5k_{n-1} - 1 = -0.5$ which is smaller than 0. Therefore, we extend the results for task sets with $p_n = p_{n-1} = \cdots = p_{n-m}$ and $p_{n-m-1} > p_{m-1}$ with the following theorem.

Theorem 4 *Given a non-preemptive harmonic task set T with $k_i = 1$ for $n - m < i < n$, $k_{n-m} \geq 2$, and $k_j \geq 1$ for $1 \leq j < n - m$, Precautious-RM is an optimal scheduling algorithm if (8) and the following conditions hold:*

$$\begin{cases} V_i^{'} > 0, & 1 < i < n \\ V_i^{'} \geq 0, & i = 1 \end{cases} \tag{11}$$

$$c_n^{'} \leq p_n \tag{12}$$

$$(p_n - c_n^{'}) \leq c_i \leq 2(p_n - c_n^{'}) \tag{13}$$

where

$$V_i^{'} = \begin{cases} k_i V_{i+1}^{'} - 1, & 1 \leq i < n - m \\ 0.5, & i = n - m \end{cases} \tag{14}$$

$$c_n^{'} = \sum_{i=n-m+1}^{n} c_i \tag{15}$$

Proof When (12) holds, it is possible to consider all tasks with $k_i = 1, n - m < i < n$ as one new task $T_n^{'} : (c_n^{'}, p_n)$. Then, by removing tasks $\{T_{n-m+1}, T_{n-m+2}, \ldots, T_n\}$ from T, adding task $T_n^{'}$ to T, and re-indexing the resulting tasks, it is easy to see that conditions of Theorem 3 will be valid, and consequently, Precautious-RM can schedule the task set. □

Although the concept of scheduling one task between two consecutive executions of T_n instances has some benefits, the system resources might be wasted because of artificial idle intervals placed at the end of tasks with execution times smaller than $2(p_n - c_n)$ as shown in Fig. 6.

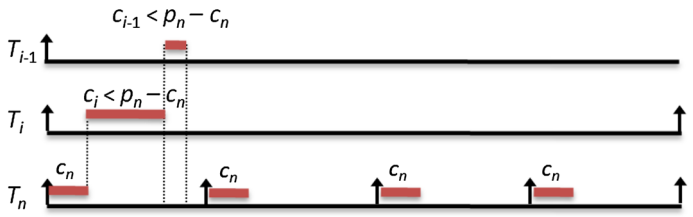

Fig. 7 An example of interference between vacant intervals because of short execution times

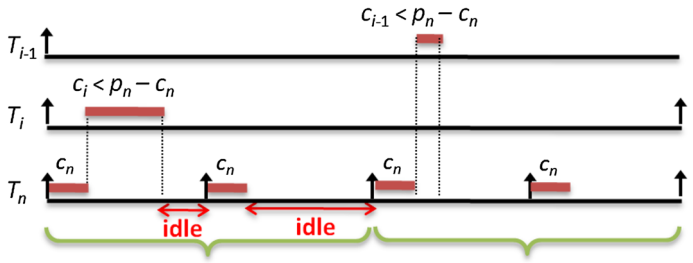

Fig. 8 Scheduling of the same tasks of Fig. 7 with LP-RM

4.4 Relaxing the lower bound condition of task execution times

For some of the theorems, we have assumed that the execution time of the tasks have to be greater than the slack of the task with the smallest period. Using this simplification, we have achieved Theorems 3 and 4 as well as Lemmas 2 and 3. However, when the tasks have naturally small execution times, e.g., the real application presented by Anssi et al. (2013) or many control applications with short execution time, this assumption might not be realistic and can limit the applicability of the theorems and lemmas.

When the execution times are small or there exist more than one task that can be scheduled along with one release of T_n (as shown in Fig. 7), vacant intervals created from two consecutive instances of T_n interfere with each other. This leads to unpredictable jitter perturbation. Throughout the rest of this section, we relax the limiting Condition (10) of Theorem 3 and Condition (13) of Theorem 4 on the execution times of the tasks. We show that using a modified version of Precautious-RM, the constructed theoretical backgrounds still remain valid. This algorithm, shown in Algorithm 2, is called Lazy-Precautious-RM (LP-RM), since it defers execution of a low-priority task, even if it does not violate deadline of the next incoming T_n. The algorithm only executes one task between two consecutive instances of T_n and it does not put other tasks in the vacant intervals.

According to LP-RM, $f(t) = 0$, if $\lfloor t/p_n \rfloor = 2h$, $h \in N$, namely $\lfloor t/p_n \rfloor$ is an even number; otherwise $f(t) = 1$. Figure 8 shows how LP-RM schedules tasks with execution times smaller than $p_n - c_n$. As it can be seen, LP-RM wastes some system processing resources but maintains more predictability.

Theorem 5 *LP-RM is optimal when it is applied on a task set consistent with the conditions of Theorems 1 or 3, even when* $c_i < p_n - c_n, 1 \leq i < n$.

Algorithm 2 Lazy-Precautious-RM

Input T, t
T is the task set, and $t \in R$ is the current time
Output S, t_{next}
S is the next task to be scheduled ($S \leftarrow null$ means that processor remains idle), and
$t_{next} \in R$ is the next time to activate the scheduler.

1: $r_n^{next} \leftarrow$ next release time of T_n according to t
2: **for** $i \leftarrow n$ **downto** 1 **do**
3: **if** (current instance of T_i has not been executed) **then**
4: **if** $((i = n)$ **or** $(t + c_i \leq r_n^{next} + p_n - c_n$ **and** the latest executed task is T_n **and** $f(t) = 0))$ **then**
5: $S \leftarrow T_i$
6: $t_{next} \leftarrow t + c_i$
7: **return** S, t_{next}
8: **else**
9: **break**
10: **end if**
11: **end if**
12: **end for**
13: $S \leftarrow null$
14: $t_{next} \leftarrow r_n^{next}$
15: **return** S, t_{next}

Proof To prove this theorem, we have to show that LP-RM schedules at most one task with the highest priority between two consecutive instances of T_n, and hence, it performs the same as Precautious-RM when the execution times of the tasks are larger than $p_n - c_n$.

According to the system model in Sect. 2, the system starts at time 0 and all tasks are released synchronously at that time which leads to $f(0) = 0$ at the first release of T_n. Thus, LP-RM first schedules T_n and then is activated again at $t = c_n$ with $f(t) = 0$. Suppose at that time, T_i is the task with the highest priority, thus it will be scheduled by LP-RM until time $c_n + c_i$. If $c_i < p_n - c_n$, we have $c_n + c_i < p_n$, and hence, the algorithm schedules an idle interval until the second release of T_n since the latest executed task is not T_n (see Line 4 of Algorithm 2). If $c_n + c_i > p_n$ the next time the algorithm will be activated is at $t = c_n + c_i$ and it is after the second release of T_n. In both cases, when the algorithm is activated, $f(t) = 1$ which means that no task except T_n is allowed to be scheduled until the release of the third T_n at $2p_n$. From that time, the aforementioned pattern of execution of the tasks will continue to happen forever. Consequently, at the release of any T_n with $f(t) = 0$, the processor will be idle, and the assumptions we made during this proof remain valid, which completes the proof. □

It is worth mentioning that conditions of Theorems 3 or 5 can be found in many of the non-preemptive real-time applications such as the one introduced in (Anssi et al. 2013). Many of these systems have tasks with very small execution times, e.g., a few milliseconds, but with harmonic and wide periods, which are around tens to hundreds of milliseconds. Also in these systems, there could be groups of cooperative tasks with the same periods. Using Precautious-RM or LP-RM, these tasks can be scheduled non-preemptively with no deadline miss.

5 Response time and jitter analysis of Precautious-RM

As mentioned before, one of the important properties of Precautious-RM is that, according to (4), it bounds WCRT of each task to a value which can be obtained in $O(1)$ at design-time. However, this bound is not tight. Throughout this section, we derive conditions to tighten this bound. Also we derive exact value of BCRT of the tasks when conditions of Theorem 3 hold. From now on, this section considers only feasible non-preemptive harmonic task sets.

Lemmas 2 and 3 to are important from different aspects; first, they can be used to determine which task encounters RTJ in Precautious-RM, second, they show that how many low priority tasks can be scheduled along with one release of task T_i, $1 \leq i < n$, and third, they can be employed to find exact WCRT of the tasks. The following corollaries illustrate results of these lemmas.

Corollary 1 *Given harmonic task set T with $\sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1, k_i = 3$, and $p_n - c_n \leq c_i \leq 2(p_n - c_n), 1 \leq i < n$, WCRT of Precautious-RM shown by (4) is tight considering:*

$$WCRT_n = c_n + 2(p_n - c_n) - max\{c_i\}_{i=1}^{n-1} \tag{16}$$

Proof According to Lemmas 2 and 3, we have $V_n = 0.5$, and $k_i = 3$, $1 \leq i < n$, thus $V_i = 0.5$. It means that each task T_i has just 0.5 units of idle intervals available for scheduling task T_{i-1}. As a result, low priority tasks will not find any other opportunity to be executed sooner than $2p_{i+1} - p_n + c_n + c_i$ as said in (4). Also it means that the best and the worst possible start-time of that task are identical (since that task has no other options to start earlier). Thus, the BCRT and the WCRT of that task are identical. For task T_n, its WCRT happens when it has to wait for finishing of the execution of a low priority task in the system, i.e., $2(p_n - c_n) - max\{c_i\}_{i=1}^{n-1}$. But its BCRT happens at its first release at time 0, and it is c_n. □

Corollary 2 *Given binary harmonic task set T consistent with conditions of Lemma 1, tight bound of WCRT of the tasks scheduled by Precautious-RM is:*

$$WCRT_i = \begin{cases} p_{i+1} - S_{i+1} + c_i, & 1 \leq i < n \\ max\{c_i - S_{i+1}\}_{i=1}^{n-1} + c_n, & i = n \end{cases} \tag{17}$$

$$BCRT_i = \begin{cases} WCRT_i, & 1 \leq i < n \\ c_n, & i = n \end{cases} \tag{18}$$

where S_i follows (5).

Proof Suppose the next release of T_{i+1} is at time $\sigma = p_{i+1}$. According to Lemma 1, when $S_{i+1} < c_i$, $1 \leq i < n$, task T_i is scheduled at $\sigma - S_{i+1}$. Also R_0-Schedule of T_{i+1} has no idle-time, thus, T_i cannot be started before $\sigma - S_{i+1}$, meaning that the $WCRT_i = BCRT_i$. However, for task T_n, $BCRT_n = c_n$ since it has to be started right after its first release. Because of Lemma 1, we know that each task T_i, $1 \leq i < n$ will be started S_{i+1} units of time before the release of T_n, and it will be finished $c_i - S_{i+1}$ units of time after the release of that T_n. In this case, the latest start-time of T_n with

respect to its release is $max\{c_i - S_{i+1}\}_{i=1}^{n-1}$, which is defined in (17). Thus the proof is complete. □

In the following, the BCRT and WCRT of the tasks in the presence of conditions of Theorem 3 will be studied.

Theorem 6 *If conditions of Theorem 3 are valid and Precautous-RM is used, the BCRT of each task $T_i, 1 \leq i < n$ happens when the processor is idle at its release time. Also the WCRT of task T_i happens if the processor is busy at its release time.*

Proof According to the definition of harmonic tasks, all tasks with priorities higher than T_i are released synchronously with the release of T_i. Consequently, T_i always has to wait for those tasks, which means that its BCRT will never be smaller than the case where the processor is idle at its release. On the other hand, in the presence of conditions of Theorem 3, if the processor is busy when T_i is released, Precautious-RM schedules the first T_n right after finishing the low priority task which has blocked the processor. Hence, half of the vacant-interval becomes full and it cannot be used to schedule other tasks. Consequently, other released tasks including T_i suffer from start-time delay which leads to occurrence of their WCRT. Since the maximum start-time delay is equal to $p_n - c_n$, the WCRT of each task will not be larger than the one that happens when the processor is busy at the release of the task. □

To find the exact WCRT of each task, we need to know the correct vacant interval in which the task is scheduled. For this aim we define the capability level of a task as:

Definition 2 Capability level of a task, denoted by $L_i, 1 \leq i < n$, is the number of tasks that can be scheduled along with the first release of T_i.

According to Lemma 3 and the Definition 1, the number of low priority tasks which can be started along with one release of $T_i, 1 \leq i < n$, is:

$$L_i = n - i + 1 + \lceil V_i \rceil. \tag{19}$$

The reason is that we have $n-i$ tasks with higher priority than T_i, T_i itself, and $\lceil V_i \rceil$ tasks with lower priority than T_i. It is worth mentioning that if V_i is not an integer, the final release of T_n along with the release of T_i will be used to create a vacant interval with the next T_n belonging to the next release of T_i. In such situation, if any task $T_j, 1 \leq j < i$ has been scheduled in that interval, we count it within the capability level of T_i. That is why instead of V_i, we have used $\lceil V_i \rceil$ in (19).

Theorem 7 *Given harmonic task set T, consistent with conditions of Lemma 3, response time of task $T_i, 1 \leq i < n$ has no jitter and it is exact when V_i is integer, i.e., $V_i = \lfloor V_i \rfloor$, or when $L_i > n$.*

Proof It is enough to show that the schedule produced by Precautious-RM for task T_i will be exactly repeated at each release of the task. In such case, if the processor is idle at the release of the task, it remains idle in the further releases as well. If V_i is an integer value, it means that there will be no task that starts its execution in one of

the vacant intervals of T_i but cannot be finished before the deadline of T_i. As a result, the processor will be idle at the next release of T_i. On the other hand, if V_i is not an integer value but $L_i > n$, we can deduce that all other low priority tasks have been scheduled during one release of task T_i because the number of vacant intervals along with T_i is more than the number of tasks in the system. □

One of the results of Theorem 7 is that if $V_i = 2h$ for $1 \leq i < n$ and $h \in \{0, 1, \ldots\}$, tasks T_j, $1 \leq j \leq i$ do not encounter any jitter in their response time. It is important to note that because of Lemma 2, the only task that starts before any other task in the system is T_n. It leads to high predictability of the system by avoiding jitters, because jitter can negatively affect the quality of service of many applications (Marti et al. 2001, 2002; Buttazzo et al. 2013).

Corollary 3 *If conditions of Theorem 3 are valid and for task T_i, $1 \leq i < n$, we have $L_i \leq n$ and $V_i \neq \lfloor V_i \rfloor$, the WCRT of that task appears in its second release.*

Proof Assume $V_i \neq \lfloor V_i \rfloor$ and $L_i \leq n$. It means that the final vacant interval of T_i will be occupied by a low priority task since its capability level is not greater than n. As a result, in the second release of T_i, the processor is certainly busy by a low priority task which has been started recently within the final vacant interval of T_i. Thus according to Theorem 6 the second release of T_i has the longest response time. □

Corollary 4 *If conditions of Theorem 3 are valid, each task has either one possible response time, or two.*

Proof According to Theorem 7, task T_i, $1 \leq i < n$ has just one value for its response time if V_i is integer or $L_i > n$. On the other hand, since the processor is either busy or idle at the release of this task, based on Theorem 6 each task has at most two possible values for its response time. □

To calculate the BCRT or WCRT of the tasks we construct an algorithm called response time seeker (RTS) presented in Algorithm 3. In this algorithm we gradually seek for a tighter bound for the requested response time. RTS algorithm can be used to find either BCRT or WCRT. The former is obtained when $type = WCRT$ and the latter is acquired when $type = BCRT$. Their difference appears only in Line 6 of Algorithm 3, where we have to determine the value of b. If $b = 1$, half of the first vacant interval is already occupied by a low priority task, thus, we decrease 0.5 unit from the available vacant intervals of the underlying task. Afterwards, RTS gradually tightens lower bound of the response time (denoted by RT_i^l) using the loop in Lines 14 to 16 of Algorithm 3. In that loop, there is a call to the lower bound updater (LBU) function shown in Algorithm 4. It is worth mentioning that initial value of RT_i^l is $c_n + c_i$ because each task will be finished after one successful execution of T_n.

LBU function gets target task T_i, integer value h, and current value of b, then it calculates the number of releases of task T_{i+1} (denoted by a) which are needed to schedule task T_i plus h number of low priority tasks. As a result, RT_i^l can be updated by value $p_{i+1}(a-1)$ as the return value of LBU function.

LBU uses two variables, $0 \leq v \leq V_{i+1}$ and $1 \leq a \leq k_i$. The former represents current available vacant intervals which can be used to schedule low priority tasks.

Algorithm 3 RTS

Input $T_i, type$
T_i: the target task,
$type$: type of the output {BCRT, WCRT}
Output RT_i
RT_i: the requested response time of T_i

1: **if** $(i = n)$ **then**
2: **return** $2(p_n - c_n) - max\{c_j\}_{j=1}^{n-1} + c_n$
3: **end if**
4: $RT_i^l \leftarrow c_n + c_i$
5: **if** $(type =$ WCRT) **and** $((L_i < n)$ **or** $(v_{i+1} \neq \lfloor v_{i+1} \rfloor))$ **then**
6: $b \leftarrow 1$
7: **else**
8: $b \leftarrow 0$
9: **end if**
10: **if** $(i = n - 1)$ **then**
11: **return** $RT_i^l + b \times p_n$
12: **end if**
13: $h \leftarrow 1$
14: **for** $j \leftarrow i$ **to** $n - 1$ **do**
15: $RT_i^l \leftarrow RT_i^l + LBU(T_i, h, b)$
16: **end for**
17: **return** RT_i^l

The latter, however, represents the smallest possible number of the releases of task T_{i+1} (we call it release-counter hereafter). Since b is either 0 or 1, $0.5b$ shows whether half of V_{i+1} is already occupied by a low priority task or not. At the start of LBU function, h has been increased because we have to guarantee that task T_i is scheduled during one release of itself. Then we consider $v \leftarrow V_{i+1} - 0.5b$, since if half of the first vacant interval is busy, we cannot use it to schedule the current h tasks.

In the while-loop (Lines 4 to 9 of Algorithm 4), LBU tries to find a proper location for the h^{th} task. If $\lceil v \rceil < h$, it means that the current release of T_{i+1} has not enough room to cover h^{th} task. Consequently, we have to move to the next release of T_{i+1} by increasing a. However, since current release of T_{i+1} has v available vacant intervals, at most $\lceil v \rceil$ out of h tasks can be scheduled during that release of T_{i+1}, thus we update h by $h \leftarrow h - \lceil v \rceil$. On the other hand, if $\lceil v \rceil \neq v$, it means that half of the final vacant interval in the current release of T_{i+1} is used, leading to the fact that the processor is busy at the next release of T_{i+1}. In this condition, we set $b = 1$ (in Line 7 of Algorithm 4) and decrease the 0.5 unit of the vacant intervals from the next V_{i+1} (shown in Line 8). While-loop of LBU stops if $\lceil v \rceil \geq h$. It means that the current release of T_{i+1} can be successfully used to schedule the h low priority tasks (including T_i itself), thus the loop has to be stopped and we have already found the proper value for a.

The number of iterations of the while-loop in Lines 4 to 9 of Algorithm 4 depends on the minimum value among n and $K^{max} = max\{k_j\}_{j=1}^{n-1}$. Suppose $n < K^{max}$; then the worst number of iterations happens when h^{th} low priority task has to be scheduled at the final release of T_{i+1} under T_i. Also considering $V_{i+1} \leq 1$, at most $O(h)$ iterations are required to find the final value of a. Moreover, if $n > K^{max}$, since

Algorithm 4 LBU

Input T_i, h, b
T_i: the target task,
h: number of the tasks that have to be scheduled,
b: { 0,1 }: start-time condition for task T_i.
$type$: type of the output {BCRT, WCRT}
Output The lower bound of the response time of T_i along with one release of T_{i+1}.
This function also updates h and b.
1: $h \leftarrow h + 1$
2: $v \leftarrow V_{i+1} - 0.5b$
3: $a \leftarrow 1$
4: **while** $(\lceil v \rceil < h)$ **do**
5: $a \leftarrow a + 1$
6: $h \leftarrow h - \lceil v \rceil$
7: $b \leftarrow \lceil v \rceil - \lfloor v \rfloor$
8: $v \leftarrow V_{i+1} - 0.5b$
9: **end while**
10: **return** $p_{i+1}(a - 1)$

it is assumed that the task set is feasible, the final vacant interval to schedule the h^{th} task has to be found within the current T_i. Thus, the maximum number of iterations will be $O(K^{max})$ since in the worst-case a has to be smaller than or equal to K^{max}. Consequently, computational complexity of LBU is $O(min\{n, K^{max}\})$, which is at most $O(n)$. Also, the maximum number of iterations of for-loop of Lines 14 to 16 of Algorithm 3 is $n - 1$, hence, the total computational complexity of RTS algorithm is $O(n^2)$.

An example of a task set scheduled by Precautious-RM is shown in Table 1 beside Fig. 9. Utilization of this task set is 1 if for each task T_i, $1 \leq i < n$, $c_i = 2(p_n - c_n)$ (because $V_1 = 0$). Thus, there will be no wasted vacant interval for T_1.

Theorem 8 *The RTS algorithm calculates exact values of the BCRT and WCRT of any task in the task set consistent with conditions of Theorem 3.*

Proof Since at each step, the LBU function counts the number of releases of T_j, $i \leq j \leq n - 1$ that their vacant intervals are completely occupied by other low priority tasks, the response time of T_i cannot be smaller than $LBU(T_j, h, b)$. Thus the final response time value of the task, denoted by RT_i^{final}, have to be in the next release of T_j, and RT_i^l always keeps lower bound of RT_i^{final}. Moreover, because of the fact that the while-loop of Lines 4 to 9 of Algorithm 4 stops only if $\lceil v \rceil < h$, the current number of vacant intervals within the respective release of task T_j is enough to schedule the remaining h tasks. It means that RT_i^{final} cannot be greater than $RT_i^l + p_j$. Thus, at each step of the for-loop of Lines 14 to 16 of Algorithm 3, the algorithm tightens the final response time. In the final step, the smallest value which will be added to RT_i^l is p_n, meaning that we have found the release time of the closest instance of task T_n after which T_i will be scheduled. □

Table 1 A sample task set with its period ratios (k_i), vacant intervals (V_i), and capability level (L_i)

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
k_i	1	1	2	1	1	3	5	-
V_i	0	1	2	1.5	2.5	3.5	1.5	0.5
L_i	8	8	8	7	7	7	4	2

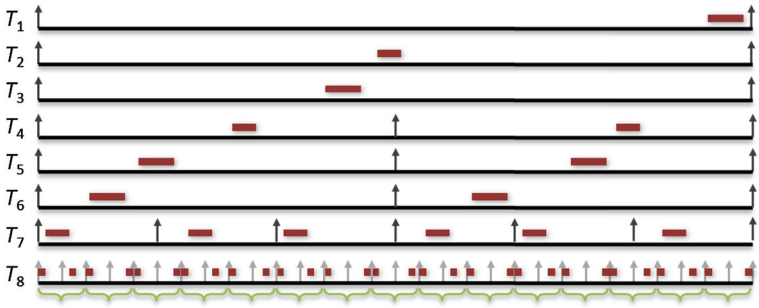

Fig. 9 The feasible task set of Table 1, scheduled by Precautious-RM (or LP-RM)

6 Applications of Precautious-RM

As mentioned before, many real-time embedded systems need predictable ways of interaction with their environment. For these systems, delays and jitters have to be bounded and preferably known at design-time. In this section, we summarize our theoretical achievements and show how Precautious-RM affects quality of service and accuracy of its target applications. Table 2 summarizes the optimality criteria of Precautious-RM with respect to the necessary conditions of schedulability. For example, Theorem 1 shows that Precautious-RM is always optimal when period ratio of the tasks is >2, and necessary conditions of schedulability of such tasks are valid. Also Theorem 2 confirms that Precautious-RM is optimal when necessary conditions of Cai and Kong (1996) hold, i.e., $c_i \leq 2S_{i+1}$ for $1 \leq i < n$, and $c_n < p_n$. But in Theorem 3, we have included sufficiency conditions for the execution time as well as vacant intervals of the tasks. It is worth mentioning that scheduling non-preemptive harmonic tasks with arbitrary period ratios has been proven to be NP-Hard by Cai and Kong (1996). Later in Theorem 5, condition $(p_n - c_n) \leq c_i$ has been relaxed but LP-RM has to be used instead of Precautious-RM.

When Precautious-RM is optimal, it has no deadline misses and response time of each task is always finite and smaller than its relative deadline. Now for the cases where Precautious-RM is optimal, Table 3 summarizes exact BCRT and WCRT of the tasks. In all of the following cases, all tasks except task T_n has either one or two possible response time values denoted by their BCRT and WCRT.

Many of the non-preemptive scheduling algorithms such as npEDF, gEDF, and cEDF may force multiple response times to each task. However, Precautious-RM guarantees that each task has at most two possible response times while all deadlines are also met. Consequently, it provides more predictable behavior for control tasks, no

Table 2 Optimality criteria of Precautious-RM with respect to necessary conditions

Harmonic property	Optimality criteria	See
$k_i \geq 3, 1 \leq i < n$	Always optimal	Theorem 1
$k_i = 2, 1 \leq i < n$	Always optimal	Theorem 2
$k_i \geq 1, 1 \leq i < n$	$(p_n - c_n) \leq c_i$ $V_1 \geq 0, V_i > 0$	Theorem 3
$k_i = 1,\ n - m < i < n$ $k_i \geq 2,\ i = n - m$ $k_i \geq 1,\ 1 \leq i < n - m$	$(p_n - c_n) \leq c_i^{'}$ $V_1^{'} \geq 0, V_i^{'} > 0$	Theorem 4
$k_i \geq 1, 1 \leq i < n$	$V_1 \geq 0, V_i \geq 0$	Theorem 5 (LP-RM)

Table 3 BCRT and WCRT of tasks scheduled by Precautious-RM

Harmonic Property	BCRT/WCRT	See
$k_i = 3, 1 \leq i < n$	Formula (4) and (16)	Corollary 1
$k_i = 2, 1 \leq i < n$	Formula (17) and (18)	Corollary 2
$k_i \geq 1, 1 \leq i < n$	RTS Algorithm	Theorem 8

deadline miss, as well as fixed BCRT and fixed WCRT. In many control applications, the stability analysis relies on the guaranteed periodic behavior (Eker et al. 2000), that is achieved when the scheduling algorithms have no deadline misses. Moreover, known sampling and I/O delays can be tolerated by the application itself (Marti et al. 2001, 2002; Nilsson et al. 2010). Accordingly, the system quality of service and accuracy can be improved by a co-design approach when the application is equipped with means to tolerate particular delays at design-time and the scheduler guarantees the same delays at run-time. This cannot be obtained if there are missed deadlines or multiple response time values are possible for the control tasks. Also it is worth mentioning that response time analysis of the other non-preemptive algorithms might have considerable time-complexity (Eisenbrand and Rothvoß 2008, 2010), while according to the RTS algorithm, response time of Precautious-RM can be calculated in $O(n^2)$.

In practice, task execution times suffer from jitter. It leads to I/O delay jitter, and consequently to RTJ. It can be propagated to the system if a number of tasks have to be scheduled one after the other with no idle time insertion. This situation happens in many work-conserving scheduling algorithms such as npRM, npEDF, and gEDF. In that case, not only I/O delay jitter of each task affects its own quality of service, but also it causes the other tasks to suffer from sampling delay jitter. This propagated jitter worsens the quality of service of the overall system. However, Precautious-RM and LP-RM stop this jitter propagation by proper use of idle intervals. In our algorithms, when conditions of Theorem 3 hold, each task has to be executed after one T_n which is executed right after its release. Thus, the only task that affects on sampling delay of other tasks is T_n because if low priority task T_i, $1 \leq i < n$ suffers from I/O delay jitter, it might only affect the sampling delay of the next incoming T_n instead of any other low priority task in the system. Accordingly, it can be seen that Precautious-RM and LP-RM use idle intervals not only to preserve schedulability, but also to decrease the

effect of jitter propagation in the system. It is worth mentioning that the assumption of knowing exact execution-time of the tasks in Sect. 2 is not necessary for optimality of Precautious-RM and LP-RM, i.e., if instead of exact execution times, the worst-case execution-time (WCET) of the tasks are given, still the proposed scheduling algorithms are optimal. As it has been proven by Theorems 1, 2, and 4, as long as the WCET of each task is smaller than or equal to $2(p_n - c_n)$, Precautious-RM is able to guarantee its deadline, while other non-optimal scheduling algorithms such as npEDF and gEDF might have missed deadlines (Jeffay et al. 1991) even when the actual execution times at run-time are smaller than the respective WCETs.

As mentioned earlier, T_n is the only task that might have more than two response time values. Besides, if there is a task with $c_i = 2(p_n - c_n)$, $1 \leq i < n$ in the system, WCRT of T_n becomes equal to its period. In fact, in most cases, Precautious-RM sacrifices quality of service of the task with the smallest period to preserve schedulability of the other tasks, however, in many of real-time applications such as control systems (Bini and Cervin 2008), quality of service and accuracy also depend on the length of the task periods. If a task has a very short period, it is able to tolerate delay and jitters by itself. Also when the period is short, delays are also short. Consequently, task T_n might not necessarily lose its accuracy because of the schedule offered by Precautious-RM.

7 Experimental results

In this section, performance of Precautious-RM (P-RM) and LP-RM are compared against non-preemptive scheduling algorithms npEDF, gEDF (Li et al. 2007), cEDF (Ekelin 2006), and GSSP (Cai and Kong 1996). Since the algorithm presented by Deogun and Kong (1986), only works for the tasks with constant period ratio >2, we have used GSSP which is an improved version of (Deogun and Kong 1986). This algorithm is claimed to be optimal when first m tasks have binary periods and the other $n - m$ tasks have period ratio that satisfies relation $\frac{p_1}{(2p_i)} \geq \sum_{j=1}^{i-1} \frac{p_1}{p_j}$, $1 < i \leq n$. It is worth mentioning that the results of npRM have been omitted in the paper because in harmonic task sets, npEDF and npRM are identical if we use period as the tie breaker for tasks with the same deadline.

The performance measures are MR, normalized average values of response time (RT) and RTJ, where the latter is the difference between WCRT and BCRT. MR is the ratio of missed jobs (missed instances of the tasks) to the total number of jobs in one hyperperiod. RT and RTJ are obtained by averaging the respective quantities of successfully scheduled jobs divided by the task period. It is noteworthy that in our experiments, I/O delay of each task, i.e., delay between its start and completion times, is constant since tasks are scheduled non-preemptively. Accordingly, RT is only affected by sampling delay, namely the delay between release and start of the task. In other words, having large RT means having large sampling delay.

To show the effectiveness of Precautious-RM, we have implemented a benchmark control application, i.e., an inverted pendulum (Byl 2012; Lam 2012), which is highly sensitive to sampling and I/O delays and has firm deadlines (Liu et al. 2003). All the experiments have been done using a simulation framework developed in discrete event simulator (DEVS-suite) (2009).

Table 4 Parameters of the inverted pendulum; obtained from Byl (2012)

Parameter	Symbol	Value	Unit
Motor torque constant	K_m	0.00767	*Nm/Amp*
Gear-box ratio	K_g	3.7	*N/A*
Motor armature resistance	R	2.6	Ω
Motor pinion radius	r	0.00635	*m*
Cart mass	m_c	0.455	*kg*
Pendulum mass	m_p	0.3	*kg*
Rotational inertia	I	0.00651	$kg\ m^2$
Half-length of pendulum	l	0.5	*m*
Maximum valid period	Period	0.07	*s*

There are two sets of experiments. In the first set, we consider feasible task sets; while in the second set, general harmonic tasks are considered which might not be schedulable even with a clairvoyant non-preemptive scheduling algorithm. For these experiments, inverted pendulum application has been used to show the effectiveness of the solution, as it is subjected to instability if the control task is missed, or sampling and I/O delays happen more than frequent or get larger during the execution of the task. Also in this system, jitters in sampling or actuation may considerably degrade the performance of control.

Each simulation experiment averages 200 simulation runs (each including 10 randomly generated tasks). The obtained results are of 95 % confidence level within ± 0.01 confidence interval.

7.1 Inverted pendulum experimental setup

In this subsection, we introduce our benchmark application which is taken from (Lam 2012). It is a state feedback controller responsible for balancing the pendulum in the upright position and it works based on a linear quadratic regulator (LQR) design using the linearized system. In an LQR design, the gain matrix K for a linear state feedback control law $u = -Kx$ is found by minimizing the following quadratic cost function

$$J = \int_0^\infty x(t)^T Q x(t) + u(t)^T R u(t) dt \tag{20}$$

where $x(t)$ is the difference in the state of the plant from the goal state, $u(t)$ is the actuation signal made by the control task, and Q and R are weighting parameters that penalize control inputs. For more details see (Lam 2012). In our experiments, we have used the inverted pendulum with the parameters listed in Table 4 which is the same as (Byl 2012). Note that in our implementation, the maximum period of the controller cannot be >0.08 s because of the stability issues.

In the experiments, we have used QoC as an indicator for steady state accuracy (SSA) (Buttazzo and Cervin 2007; Bini and Cervin 2008; Nasri and Kargahi 2012).

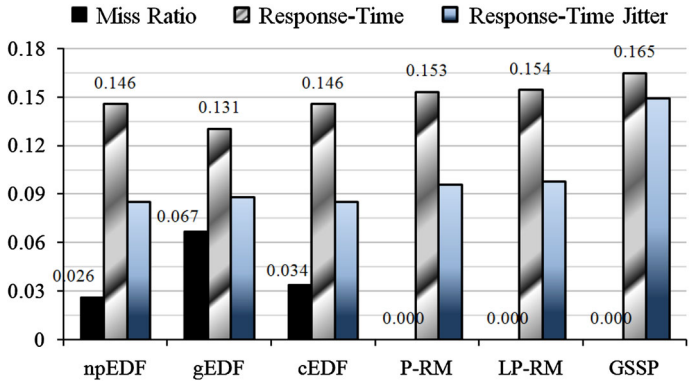

Fig. 10 MR, RT, and RTJ of the task sets satisfying Theorem 1

This measure which is usually shown by J in Eq. (20) contains the amount of error (by $x(t)^T Qx(t)$) and the energy costs (by $u(t)^T Ru(t)$) of the controller over the time. Since smaller values of J means better QoC, we have used $1/J$ and normalized that value with respect to Precautious-RM's result. It is noteworthy that in practice, in order to normalize J, the designer decision is needed to possibly obtain the maximum acceptable value for J. However, if the plant becomes unstable due to multiple job deadline misses, J tends to infinity. Therefore, to fairly compare SSA of the algorithms, we have used J of stable tasks.

For each experiment, first we have generated random tasks with arbitrarily large periods or execution times, and then we mapped the periods of the produced task set into the valid period range of the inverted pendulum application. For this aim, periods have been normalized so that $p_n = 0.02$ and $p_1 = 0.07$, and for every other task $T_i, 1 < i < n$, $p_i^{valid} = 0.02 + p_i(0.07 - 0.02)/p_1$. Also, the execution times are normalized such that $c_i^{valid} = \frac{c_i}{p_i} p_i^{valid}$.

7.2 Feasible harmonic tasks

In this section, we have considered feasible harmonic tasks satisfying conditions of Theorems 1, 2 and 5. To generate feasible tasks consistent with Theorem 1, first we have chosen random values $p_n = U[1, 10]$ and $c_n = U[0.001, 0.999]$ with uniform chance. Afterward, $k_i, 1 \leq i < n$ has been selected randomly from $\{3, 4, \ldots, 7\}$. Other periods are then constructed as $p_i = k_i p_{i+1}$. Execution times of the tasks have been selected randomly as $c_i = U[0, 2(p_n - c_n)]$, $1 \leq i < n$. Figure 10 presents MR, RT, and RTJ, and Fig. 11 shows the SSA of different scheduling algorithms.

According to Fig. 10, Precautious-RM, LP-RM and GSSP have zero MR as expected. However, their RT and RTJ are higher than npEDF, gEDF and cEDF because in many cases, they push tasks with larger periods to be executed later in the schedule. In fact, having greater RT and RTJ is the cost which is paid by Precautious-RM, LP-RM and GSSP to have zero MR. On the other hand, Precautious-RM has less RT

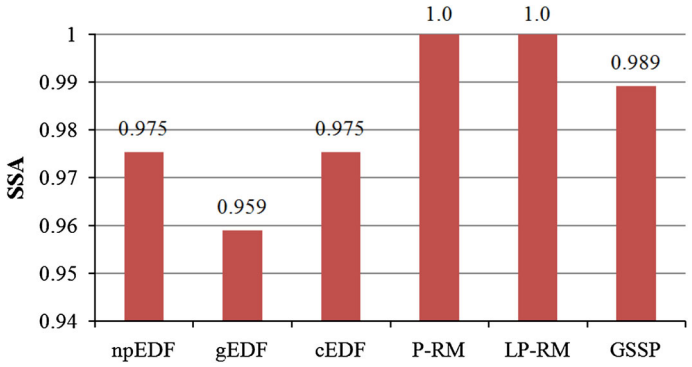

Fig. 11 SSA of the task sets satisfying Theorem 1

and RTJ in comparison with LP-RM and GSSP because it uses any opportunity to schedule low priority tasks before release of the next T_n. Also since GSSP has a strict pattern to schedule the tasks, it never uses available slacks in a hasty way, thus it has the worst RT and RTJ among the other optimal solutions.

gEDF prioritizes a task with smaller execution time among a group of tasks with almost the same deadline. Thus RT is diminished at the cost of increased MR. cEDF is non-work-conserving and it inserts some idle times to protect schedulability of the urgent tasks which will be released later. As shown in Fig. 10, RT of cEDF is higher than gEDF and npEDF because this algorithm is deceived by the structure of periods of harmonic tasks. According to (Ekelin 2006), cEDF defers the execution interval of a job if it is overlapped with another urgent job. Unfortunately, in a set of harmonic tasks, this decision leads to continually pushing jobs with larger periods later on the schedule. To optimally schedule harmonic tasks, it is required that sometimes a task with the smallest period is pushed later to create an opportunity to execute jobs with larger periods (as shown in Figs. 1, 2).

As shown in Fig. 11, Precautious-RM and LP-RM have the highest SSAs. This figure together with Fig. 10 shows the importance of MR as well as RT and RTJ in improving SSA. Although MR of GSSP is zero, because of its large RT and RTJ, its accuracy is less than Precautious-RM. Moreover, EDF-like algorithms, i.e., npEDF, gEDF and cEDF have smaller SSA because of their MR. Particularly, it can be confirmed from SSA of gEDF with respect to cEDF and npEDF that the lower RT but higher MR leads to low SSA.

Computational costs of the scheduling algorithms used in our experiments are different. It is worth mentioning that cEDF needs considerable amount of memory and has long look-up times when it is applied on a set of harmonic tasks (because it needs to maintain information about all future releases of the jobs during one hyperperiod). In a set of harmonic tasks, the number of jobs in a hyperperiod can be dramatically large (Cai and Kong 1996), thus cEDF may not be a practical algorithm. On the other hand, gEDF, npEDF, Precautious-RM, and LP-RM are naturally online algorithms with almost polynomial-time computational complexity. Moreover, GSSP algorithm presented by Cai and Kong (1996) is of $O(2^n)$ if it is used for a task set with period

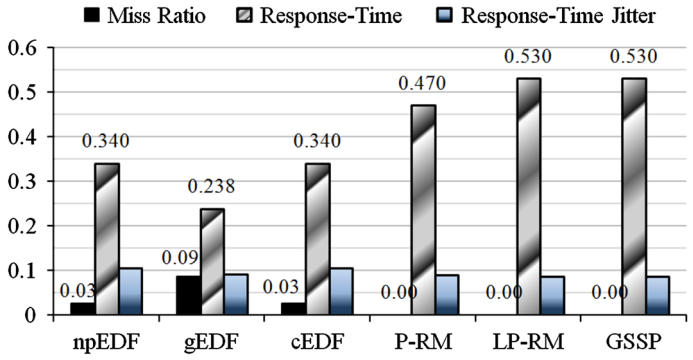

Fig. 12 MR, RT, and RTJ of the task sets satisfying Theorem 2

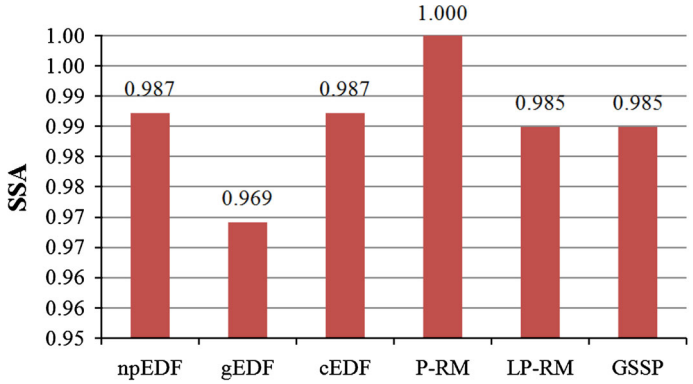

Fig. 13 SSA of the task sets satisfying Theorem2

ratio 2. By the raise in the period ratios, computational complexity of the algorithm increases exponentially.

In the second experiment, we have applied conditions of Theorem 2, i.e., we have considered $k_i = 2$, and $c_i \leq 2S_{i+1}$, $1 \leq i < n$ where S_i follows (5). To generate these tasks, random values have been selected for p_n and c_n, then $c_i = U[0, 2S_{i+1}]$, $1 \leq i < n$. Figures 12 and 13 present MR, RT, RTJ and SSA of the algorithms. As shown in these figures, Precautious-RM, LP-RM, and GSSP are optimal and have zero MR, but they have considerable RT (which is more than 40 % in average). The reason is that the optimal schedule of a binary task set usually puts each task in the middle of its period, as shown by Figs. 3 and 4. Thus, normalized RT of an optimal schedule will be around 50 % of the period. The small RTJ of the algorithms shows that in most of the cases, the BCRT and WCRT of the tasks were close to each other. Among the optimal algorithms Precautious-RM, LP-RM, and GSSP, Precautious-RM with the smallest RT, has the highest SSA in comparison with Fig. 10. The reason that SSA of LP-RM is lower than Precautious-RM is that it does not use possible options to schedule more than one task between two consecutive instances of T_n. On the other hand, among EDF-like algorithms gEDF with the highest MR has the lowest SSA.

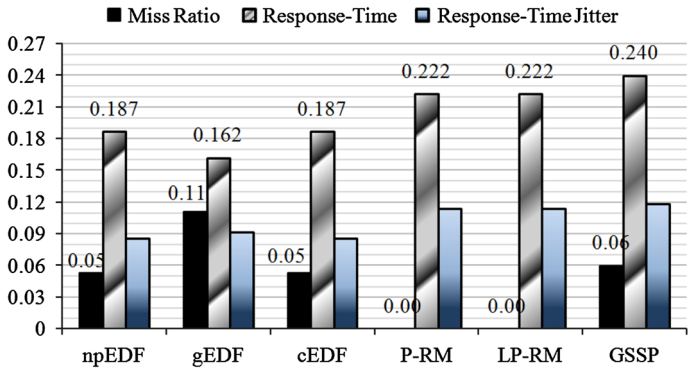

Fig. 14 MR, RT, and RTJ of the task sets satisfying Theorem 5

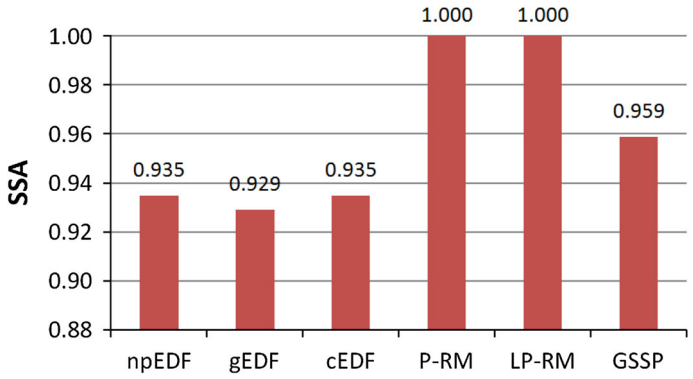

Fig. 15 SSA of the task sets satisfying Theorem 5

Although npEDF and cEDF have some MR, their SSA is higher than LP-RM and GSSP with no MR, because those algorithms has the highest RT. However, SSA of none of these algorithms is as good as Precautious-RM.

Finally, in the third experiment, we have applied conditions of Theorem 5. To generate these tasks, random values have been selected for p_n and c_n, then $c_i = U[0, 2(p_n - c_n)]$, and $k_i = \{1, 2, \ldots, 7\}$, $1 \leq i < n$. Afterwards, we have evaluated feasibility of the task set using conditions of Theorem 5 (and Theorem 3). If the generated task set has not been feasible, another task set has been generated until a feasible set is achieved for the run.

Figures 14 and 15 illustrate MR, RT, RTJ and SSA of the algorithms. According to these figures, GSSP is not optimal when period ratios can be greater than or equal to 1. Further, it has large RT which leads to smaller SSA in comparison with Precautious-RM and LP-RM. Even it can be seen that SSA of npEDF and cEDF are close to SSA of GSSP. On the other hand, LP-RM has grater response time with respect to Precautious-RM. It is because LP-RM schedules one task between two consecutive executions of T_n, as described in Sect. 4.4.

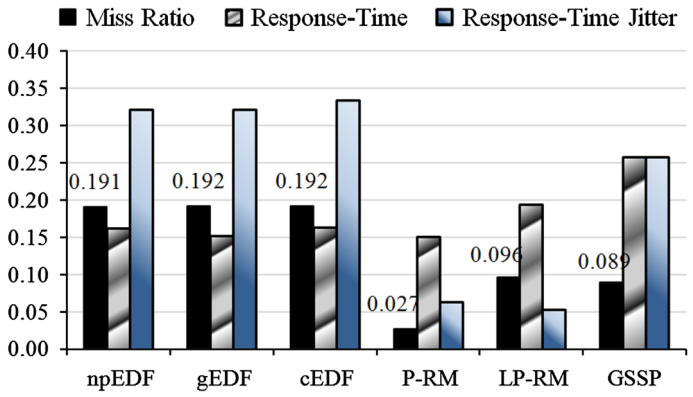

Fig. 16 MR, RT, and RTJ of general harmonic tasks

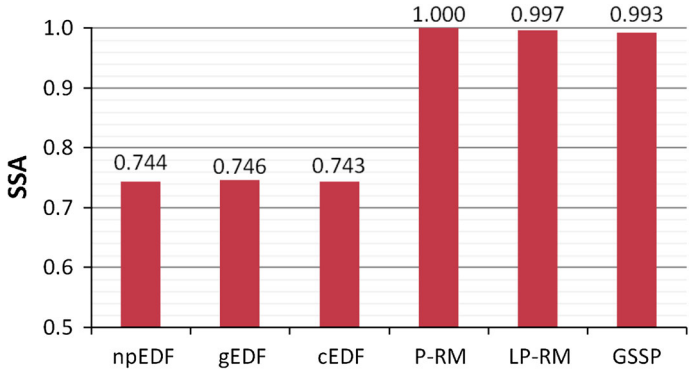

Fig. 17 SSA of general harmonic tasks

7.3 General harmonic tasks

In this section, we consider general harmonic tasks with arbitrary period ratios and arbitrary utilizations. To generate these tasks we have considered different utilizations from 0.1 to 1.0. For each utilization $u_{current}$, we have created n random values between [0, 1], then these values have been normalized such that sum of them becomes equal to $u_{current}$. For each task, k_i has been selected randomly from $\{1, 2, \ldots, 7\}$. Afterward, p_n has been chosen randomly between 0 and 10. According to the value of p_n, periods of the other tasks have been constructed, and finally, execution times of the tasks have been calculated through multiplying their utilization by their period. MR, RT, RTJ and SSA of this experiment have been illustrated in Figs. 16 and 17.

As shown in the figures, all of the algorithms have MR >0; none of them are optimal. However, Precautious-RM has the largest SSA since it has small RT as well as small MR. As expected, GSSP has grater RT than Precautious-RM and LP-RM because of special pattern that it uses to schedule the tasks. The fact laid behind significant efficiency of Precautious-RM is that it cares about the schedule of the incoming T_n. Moreover, EDF-like algorithms have considerable amounts of MR and RTJ.

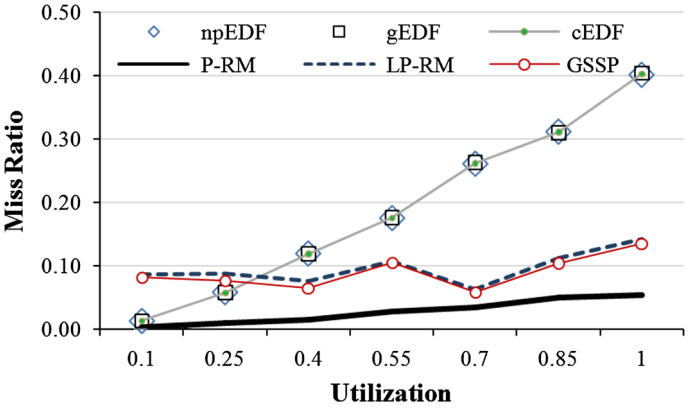

Fig. 18 MR of the algorithms in different utilizations (npEDF, gEDF and cEDF are overlapped)

As shown in the previous subsection on feasible task sets, RT of EDF-like algorithms is in average less than Precautious-RM, while in Fig. 16 it is more. The first reason is that, in general task sets, there might be infeasible tasks with execution times larger than $2(p_n - c_c)$. Since EDF-like algorithms do not take further possible deadline misses into account, they may schedule such long tasks, which not only cause deadline misses for T_n, but also may postpone the start time of the other tasks. However since Precautious-RM always cares about T_n, it will not admit execution of such long tasks. The second reason is that, when Theorem 1 or 3 are valid, there may be a lot of empty slots in the task set which can be used by EDF-like algorithms to schedule tasks earlier than Precautious-RM. For example, when $k_i = 5$ or 6, several low priority tasks can be scheduled by EDF-like algorithms along with one release of T_i. However, in general task sets, such good opportunities may not be available, thus, this greedy behavior of EDF-like algorithms results in more MR and higher RT and RTJ.

Now to study the effect of MR and RTJ on the SSA with more details, we present Figs. 18, 19 and 20. By the increase in the utilization of the task set, MR of EDF-like algorithms increases significantly while Precautious-RM, LP-RM and GSSP are able to keep their MR below (or around) 10 % with small growth. Lower SSA for EDF-like algorithms is the result of their larger MR.

As shown in Fig. 18 together with Fig. 20, pattern of decrease in SSA is related to the growth of MR. Accordingly, we see that in our benchmark application, the rate of the growth of MR in EDF-like algorithms (see Fig. 18) is higher than the rate of the diminish of the SSA (see Fig. 20). On the other hand, by the increase in MR (see Fig. 18), RT of EDF-like algorithms decreases. The reason is that when a job misses its deadline, it is thrown away of the system (due to the firm real-time nature of the system). Then, an opportunity will be produced for other tasks to be executed instead of the missed one. Consequently, those tasks may have shorter response time than usual. However, in that case, the difference between the BCRT and WCRT of such tasks increases, leading to larger amounts of RTJ.

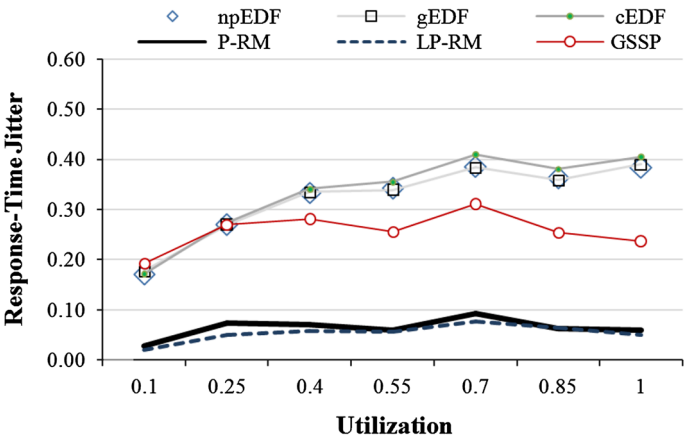

Fig. 19 RTJ of the algorithms in different utilizations

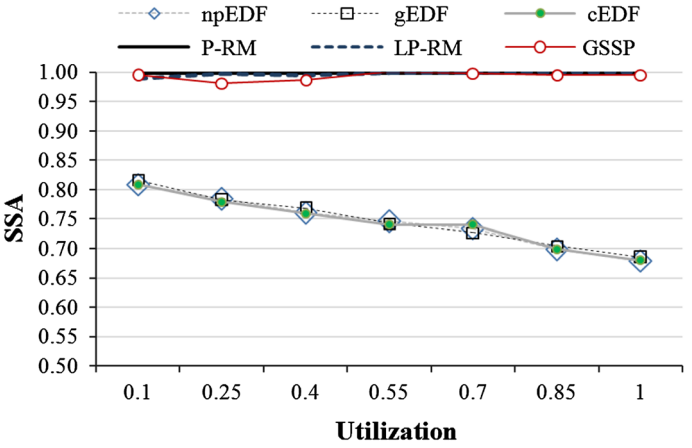

Fig. 20 SSA of the algorithms in different utilizations

According to Figs. 16 and 19, RTJ of LP-RM and GSSP are too different while their SSA is close to each other. The reason is because of the smaller MR of GSSP. In other words, when MR is not too large, RTJ might have lower impacts on SSA of the inverted pendulum.

When a task has execution time larger than two times of the slack of the task with the smallest period, it will never get a chance to be scheduled by Precautious-RM and LP-RM. This condition can be easily verified before using the algorithm. Indeed, these algorithms cannot be used to schedule tasks that do not satisfy (2).

On the other hand, if the number of missed jobs increases drastically for a task, as it happened in EDF-like algorithms when the system utilization grows, the task might become unstable (in our case, the pendulum falls and it cannot come back to the equilibrium state). As mentioned before, we have only reported SSA of stable tasks, however, in the current experiment, because of large MR of EDF-like algorithms, and

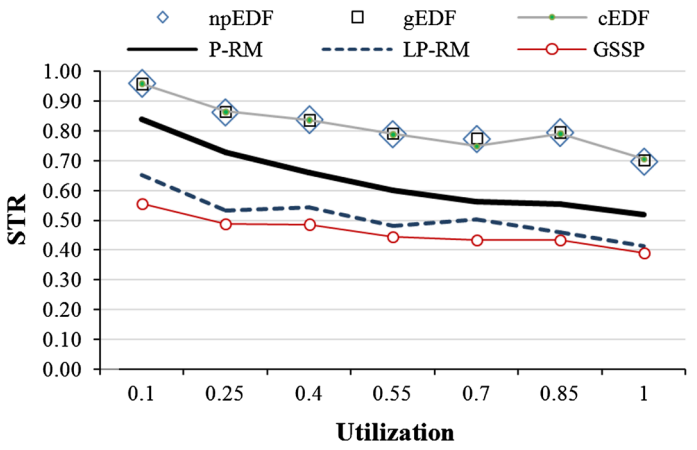

Fig. 21 STR of the algorithms

also existence of tasks that could not be scheduled by Precautious-RM, we report another measure called stable task ratio (STR) which is the ratio of stable tasks to the total number of tasks. Figure 21 shows STR of the algorithms. As said before, decrease in STR of EDF-like algorithms is because of the increase in their missed jobs which leads to unstable tasks. However they have relatively more stable tasks than Precautious-RM, LP-RM and GSSP.

Figure 21 together with Fig. 18 shows that although Precautious-RM has small MR (3 % in average), it has about 65 % STR. It means that about 35 % of the tasks has not been scheduled properly and they become unstable. At the same time we can see that these tasks have only 3 % of the jobs in the system. In other words, most of the tasks that are not scheduled by Precautious-RM are those with small number of releases. Note that if a task has very long execution time, it inevitably causes deadline misses for other tasks. That is why MR of EDF-like algorithms is significant. Moreover, since LP-RM is more conservative than Precautious-RM, it has even smaller STR because it only schedules one task between two consecutive instances of T_n. However, GSSP is still worse than LP-RM because it cannot schedule tasks with the same periods.

As mentioned previously, when conditions of Theorem 3 hold, Precautious-RM guarantees that each task has either one response time or two. But in the current experiment, Theorem 3 does not hold. Figure 22 shows the average response time diversity (RTD) of the tasks. By RTD we mean average number of response times that a task might encounter during one hyperperiod. As it can be seen, Precautious-RM and LP-RM could maintain RTD below 1.12 even in general harmonic tasks. It means that most of the tasks have just one or a small number of response times (it is also consistent with Fig. 19). RTD of GSSP is about 1.45 in average and it is relatively close to our algorithms because of the similarity of the structure of the produced schedule; they also use idle times, and put each task between two consecutive execution of T_n. On the other hand, EDF-like algorithms have greater RTD which is also growing by the growth of the utilization. Among them, gEDF has the smallest and cEDF has

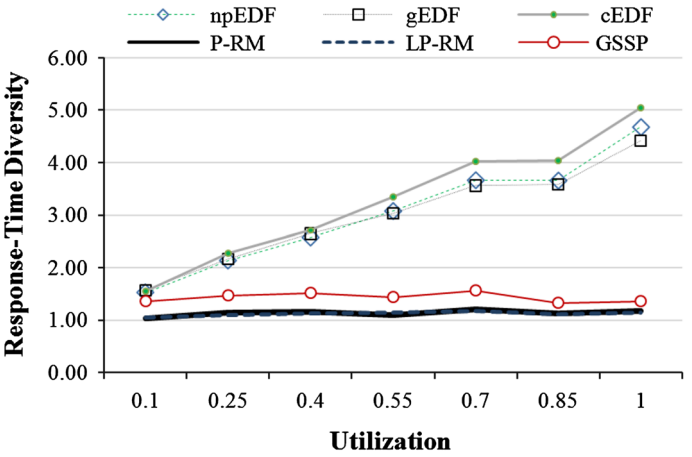

Fig. 22 response time diversity of the tasks scheduled by different scheduling algorithms

the largest RTD. The reason might lay on the fact that, in gEDF, tasks with shorter execution times are preferred over longer tasks, and hence, they have usually the same pattern of schedule and less RTD. However, npEDF is more sensitive to the missed jobs. When a job is missed, it schedules another one which is ready at the moment. It leads to more jitter in the response time. Thus, in general, RTJ and RTD of the tasks increase for EDF-like algorithms.

8 Conclusions

In this paper, we have presented Precautious-RM as a non-preemptive linear-time online scheduling algorithm for real-time harmonic tasks. We have studied optimality of Precautious-RM in three cases: for tasks with period ratio >2, for binary tasks with period ratio 2, and for a general case when there might be tasks with the same period in the system. In the latter case, we have presented sufficient conditions of schedulability; first by limiting the execution times to be greater than the slack of the task with the smallest period, and then by removing this restrictive assumption using our next scheduling algorithm called LP-RM. Later we have derived tight bounds for the best- and the worst-case response time of the feasible tasks. It is also shown that using Precautious-RM and LP-RM, each task has either one or two possible response times (and sampling delays). We obtained conditions in which a task has no RTJ. Experimental results are also presented to show the effectiveness of these algorithms in decreasing MR, response time, and RTJ in comparison with several non-preemptive scheduling algorithms. Also, we have shown that even when the task set is not feasible, yet Precautious-RM is efficient and has lower MR and response time jitter. Experiments on some benchmark control applications show that Precautious-RM can efficiently improve the QoC of such applications.

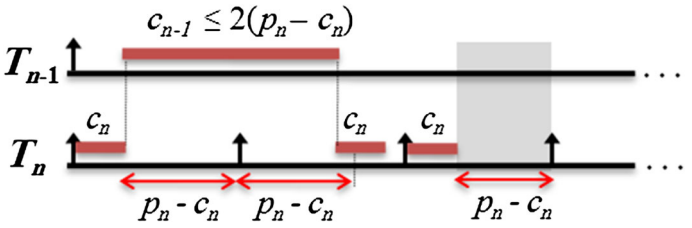

Fig. 23 Schedule of two tasks T_n and T_{n-1}, when the processor is idle

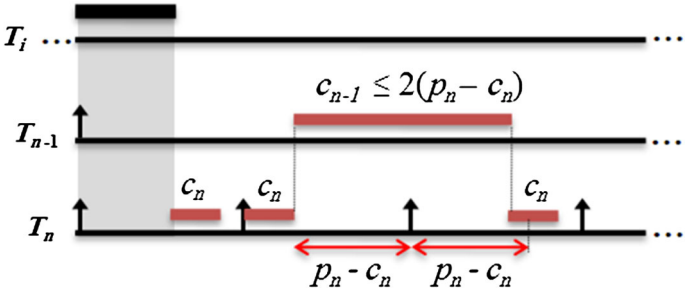

Fig. 24 Schedule of two tasks T_n and T_{n-1}, when the processor is busy

Appendix

Lemma 4 *Suppose a non-preemptive task set T consistent with Conditions (1), (2) and (3); the worst-case response time of task T_n follows (4). For task T_{n-1}, if the processor is idle at the release of T_{n-1}, its response time will be $c_n + c_{n-1}$ according to Precautious-RM, and also there will be an empty interval with length $p_n - c_n$ before the third release of T_n. Otherwise, if at the release of task T_{n-1}, the processor is busy by task T_i, $1 \le i < n-1$, the response time of T_{n-1} will be bounded by $p_n + c_n + c_{n-1}$.*

Proof As shown in Line 4 of Algorithm 1, each task is scheduled only if it does not cause the next release of task T_n to miss its deadline. Hence, none of the instances of T_n will miss their deadline because of blocking caused by a low priority task. However, the WCRT of T_n is bounded by its period since in the cases where the execution time of a task is equal to $2(p_n - c_n)$, the latest T_n has to be scheduled at the end of its period.

According to the definition of harmonic tasks in Sect. 2, on release of T_{n-1}, all other tasks with periods smaller than T_{n-1} are also released. In our case, one instance of T_n is released simultaneously with T_{n-1}. We call it the first T_n. If the processor is idle at the release time of T_{n-1} (as shown in Fig. 23), this task can be scheduled right after the first T_n and its response time will be $c_n + c_{n-1}$ which is definitely smaller than (4). Besides, since it has been executed before the third release of task T_n, then an empty interval with length $p_n - c_n$ will be created after finish of the third T_n.

Moreover, if at the release time of T_{n-1}, the processor is busy by task T_i, $1 \le i < n - 1$, when it becomes idle again, Precautious-RM will select the first T_n, since it has the highest priority. This situation has been shown in Fig. 24. In the worst-case,

when the first T_n is finished, T_{n-1} cannot fit into the remaining interval according to Line 4 of Algorithm 1; consequently, an idle time will be inserted until release of the second T_n. At that time, the algorithm will select the second T_n but when it is finished at time $p_n + c_n$, the execution of T_{n-1} is definitely started because according to Line 4 of the algorithm, the previous executed task has been T_n. In this case, the response time of T_{n-1} will be $p_n + c_n + c_{n-1}$ which is smaller than $3p_n - c_n$ since $p_n + c_n + c_i \leq p_n + c_n + 2(p_n - c_n) \leq 3p_n - c_n$. Thus, the proof is complete. □

References

Al-Sheikh A, Brun O, Hladik PE, Prabhu BJ (2011) A best-response algorithm for multiprocessor periodic scheduling. In: Euromicro Conference on Real-Time Systems (ECRTS), pp 228–237

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

Anssi S, Kuntz S, Gérard S, Terrier F (2013) On the gap between schedulability tests and an automotive task model. J Syst 59(6):341–350

Bastoni A (2010) Cache-related preemption and migration delays: empirical approximation and impact on schedulability. In: International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), pp 33–44

Bate I, Burns A, McDermid J, Vickers A (1996) Towards a fixed priority scheduler for an aircraft application. In: Euromicro Workshop on Real-Time Systems (EWRTS), pp 34–39

Bini E, Cervin A (2008) Delay-aware period assignment in control systems. In: IEEE Real-Time Systems Symposium (RTSS), pp 291–300

Busquets-Mataix JV, Serrano JJ, Ors R, Gil P, Wellings A (1996) Using harmonic task-sets to increase the schedulable utilization of cache-based preemptive real-time systems. In: International Workshop on Real-Time Computing Systems and Applications (RTCSA), pp 195–202

Buttazzo GC, Cervin A (2007) Comparative assessment and evaluation of jitter control methods. In: International Conference on Real-Time and Network Systems, pp 163–172

Buttazzo GC, Bertogna M, Yao G (2013) Limited preemptive scheduling for real-time systems: a survey. IEEE Trans Ind Inf 9(1):3–15

Byl K (2012) Balancing the inverted pendulum. Tech. rep., University of California, Santa Barbara. http://www.ece.ucsb.edu/courses/ECE147/147B_W12Byl/lab/lab3_v2.pdf. Accessed 28 May 2014

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

Davis R, Burns A, Bril R, Lukkien J (2007) Controller area network (CAN) schedulability analysis: refuted, revisited and revised. Real-Time Syst 35(3):239–272

Deogun JS, Kong MC (1986) On periodic scheduling of time-critical tasks. In: IFIP World Computer Congress, pp 791–796

DEVS (2014) Discrete-event system simulator (DEVS-suite). Arizona Center of Integrative Modeling and Simulation, Arizona State University. http://acims.asu.edu/software/devs-suite. Accessed 28 May 2014

Eisenbrand F, RothvoßT (2008) Static-priority real-time scheduling: response time computation is NP-hard. In: IEEE Real-Time Systems Symposium (RTSS), pp 397–406

Eisenbrand F, Rothvoß T (2010) EDF-schedulability of synchronous periodic task systems is coNP-hard. In: Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) Philadelphia, pp 1029–1034

Ekelin C (2006) Clairvoyant non-preemptive EDF scheduling. In: Euromicro Conference on Real-Time Systems (ECRTS) pp 23–32

Eker J, Hagander P, Arzen KE (2000) A feedback scheduler for real-time control tasks. Control Eng Pract 8(12):1369–1378

Fu Y, Kottenstette N, Chen Y, Lu C, Koutsoukos XD, Wang H (2010) Feedback thermal control for real-time systems. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 111–120

Georges L, Mühlethaler P, Rivierre N (2000) A few results on non-preemptive real-time scheduling. Tech. rep., INRIA Research, Report nRR3926

Han CC, Tyan HY (1997) A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In: IEEE Real-Time Systems Symposium (RTSS), pp 36–45
Jeffay K, Stanat DF, Martel CU (1991) On non-preemptive scheduling of period and sporadic tasks. In: IEEE Real-Time Systems Symposium (RTSS), pp 129–139
Kuo TW, Mok A (1991) Load adjustment in adaptive real-time systems. In: IEEE Real-Time Systems Symposium (RTSS), pp 160–170
Lam J (2012) Control of an inverted pendulum. Tech. rep., University of California, Santa Barbara. www.ece.ucsb.edu/roy/student_projects/Johnny_Lam_report_238.pdf. Accessed 28 May 2014
Li H, Sweeney J, Ramamritham K, Grupen R, Shenoy P (2003) Real-time support for mobile robotics. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 10–18
Li W, Kavi K, Akl R (2007) A non-preemptive scheduling algorithm for soft real-time systems. Comput Electr Eng 33(1):12–29
Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61
Liu D, Hu XS, Lemmon MD, Ling Q (2003) Firm real-time system scheduling based on a novel QoS constraint. In: IEEE Real-Time Systems Symposium (RTSS), pp 320–333
Marouf M, Sorel Y (2010) Schedulability conditions for non-preemptive hard real-time tasks with strict period schedulability analysis. In: International Conference on Real-Time and Network Systems, pp 50–58
Marti P, Fuertes JM, Fohler G, Ramamritham K (2001) Jitter compensation for real-time control systems. In: IEEE Real-Time Systems Symposium (RTSS), pp 39–48
Marti P, Fuertes JM, Fohler G, Ramamritham K (2002) Improving quality-of-control using flexible timing constraints: Metric and scheduling issues. In: IEEE Real-Time Systems Symposium (RTSS), pp 91–100
Nasri M, Kargahi M (2012) A method for improving delay-sensitive accuracy in real-time embedded systems. In: IEEE International Conference on Embedded and RealTime Computing Systems and Applications (RTCSA), pp 378–387
Nasri M, Kargahi M, Mohaqeqi M (2012) Scheduling of accuracy-constrained real-time systems in dynamic environments. IEEE Embed Syst Lett 4(3):61–64
Nawrocki JR, Czajka A, Complak W (1998) Scheduling cyclic tasks with binary periods. Inf Process Lett 65(4):173–178
Nilsson JO, Skog I, Handel P (2010) Joint state and measurement time-delay estimation of nonlinear state space systems. In: International Conference on Information Sciences Signal Processing and their Applications (ISSPA), pp 324–328
Park M (2007) Non-preemptive fixed priority scheduling of hard real-time periodic tasks. In: International Conference on Computational Science, pp 881–888
Piaggio M, Sgorbissa A, Zaccaria R (2000) Preemptive versus non-preemptive real-time scheduling in intelligent mobile robotics. J Exp Theor Artif Intell 12(2):235–245
Shih CS, Gopalakrishnan S, Ganti P, Caccamo M, Sha L (2003) Scheduling real-time dwells using tasks with synthetic periods. In: IEEE Real-Time Systems Symposium (RTSS), pp 210–219
Wu Y, Bertogna M (2009) Improving task responsiveness with limited preemptions. In: IEEE Emerging Technologies and Factory Automation (EFTA), pp 1–8

Mitra Nasri is currently a visiting researcher in Chair of Real-Time Systems at Technical University of Kaiserslautern, Germany. She received her B.S. and M.S. degrees in computer engineering from Iran University of Science and Technology in 2006 and 2008, and she is currently a Ph.D. student in the School of Electrical and Computer Engineering at the University of Tehran, Iran. Her research interests include accuracy-aware scheduling, schedulability analysis and resource management in real-time embedded systems. She is a student member of IEEE.

Mehdi Kargahi is currently an associate professor in the Department of Electrical and Computer Engineering at the University of Tehran, Iran where he joined first as an assistant professor in 2007. He received his BS degree in computer engineering from Amir-Kabir University of Technology (Tehran Poly-Techniques) in 1998 and the MS and PhD degrees in computer engineering from Sharif University of Technology in Tehran in 2001 and 2006, respectively. He has also been a researcher in the School of Computer Science at the Institute for Studies in Theoretical Physics and Mathematics (IPM) from 2003. His research interests include performance modeling and resource management of dependable and distributed real-time systems. He is a senior member of the IEEE and the IEEE Computer Society.