Non-Work-Conserving Non-Preemptive Scheduling: Motivations, Challenges, and Potential Solutions

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

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

***Abstract*—In many real-time systems, preemption is either impossible or prohibitively expensive. The problem of scheduling non-preemptive periodic tasks with known release offsets is known to be NP-Hard. In this paper, we investigate the existing non-preemptive scheduling algorithms in both categories of work-conserving and non-work-conserving algorithms, where in the former, the processing resource is not allowed to be idle as long as there is an unfinished job in the system. While describing the advantages and weaknesses of the existing scheduling solutions we show that using online non-work-conserving algorithms it is possible to schedule more task sets. In our work, we discuss the challenges to design the idle-time insertion policy (IIP) which can be combined with the existing scheduling policies such as the earliest deadline first (EDF), rate monotonic (RM), etc. Further we present a tighter necessary condition for schedulability of non-preemptive tasks. We also provide an IIP for EDF based on looking into a number of jobs in future. Through the experiments we show that the our IIP for EDF significantly increases the schedulability of non-preemptive tasks, particularly in periodic task sets. While our schedulability ratio is more than 80%, the state of the art work-conserving algorithms are about 15%.**

***Keywords*-non-preemptive scheduling; non-work-conserving scheduling; idle-time insertion policy; hard real-time systems;**

I. Introduction

In many real-time systems where there is communications over shared medium, e.g., CAN networks, preemption is either impossible or prohibitively expensive [1], [2], [3]. If tasks are not running preemptively, it is easier to guarantee mutual exclusion and access to the shared resources in the operating systems because there is no need for special resource sharing policies, and hence, it reduces run-time and design-time overheads [4], [5]. Moreover, the estimation of the worst-case execution time (WCET) becomes more accurate and less pessimistic because the locality of data in the cache or CPU pipelines will not be disturbed by the execution of other tasks [6], hence, the actual execution time will be close to the WCET measured in isolation. Besides, non-preemptive execution increases the quality of service in the control systems because it efficiently reduces the delay between sampling and actuation.

It has been shown that deciding about feasibility of a non-preemptive periodic task set with implicit deadline and known release offset is an NP-Hard [7] problem even if the periods are harmonic [8], i.e., each period divides all smaller periods. The reason is that there is no known optimal policy to decide about a feasible sequence of jobs which respects all deadlines. To find an optimal schedule, usually an exhaustive search over all possible job orderings is needed which has $O(N!)$ computational complexity [9] where N is the number of jobs in the system.

There have been several online scheduling algorithms for non-preemptive systems, among them we refer to non-preemptive EDF (NP-EDF), non-preemptive rate monotonic (NP-RM), non-preemptive fixed-priority (NP-FP) [10], group-based EDF (Gr-EDF) [11], and Precautious-RM [12], [13]. In 1991, Jeffay et al., [7] have shown that NP-EDF is optimal in the class of *work-conserving* scheduling algorithms for periodic task sets. A work-conserving algorithm never leaves the processing unit idle as long as there is an unfinished job in the system. In addition, Jeffay et al., have introduced a pseudo-polynomial time schedulability test to verify the necessary and sufficient schedulability condition for NP-EDF. However, in our paper we show that NP-EDF is not optimal in the class of work-conserving scheduling algorithms if the release offsets of the tasks are known or are zero.

Recently, with the introduction of Precautious-RM which is an online non-work-conserving scheduling algorithm based on RM [12], [13], it has been shown that it is possible to increase the schedulability of non-preemptive task sets by inserting idle intervals in the schedule. In Precautious-RM, the highest priority task with a pending job is allowed to be schedule only if it will not cause a deadline miss for the *next job* of the task with the smallest period, denoted by τ_1. If this condition is not met, an idle interval will be scheduled until the next release of τ_1. With this very simple *idle-time insertion policy* (IIP), Nasri et al., have shown that the problem of scheduling non-preemptive harmonic tasks with period ratio larger than 2 (or equal to 2), is not NP-Hard. Instead, it is solvable in polynomial-time since Precautious-RM is able to successfully schedule such task sets. These results highlight the fact that the hardness of the problem of scheduling periodic task sets with known release offset depends on the period ratio of the tasks. Besides, it shows that the IIP can be merged with the existing scheduling policies such as EDF, RM, FP, etc., in order to construct more efficient non-work-conserving scheduling algorithms.

In this paper, we investigate the existing scheduling solutions in both categories of work-conserving and non-work-conserving algorithms. We discuss the advantages and weaknesses of these solutions in order to provide

a better understanding about the potential works which can be done in the field of non-preemptive scheduling. As the second contribution of the paper, we revisit the existing necessary schedulability condition and make it tighter, particularly for task sets with period ratio smaller than 3 (where the hardness of the problem lies). As the third contribution, we introduce an IIP based on the new necessary schedulability test. We combine this policy with EDF to build our new *critical time window*-based EDF scheduling policy (CW-EDF). In this algorithm, a high priority job is allowed to be scheduled only if it will not cause a deadline miss for the *next job* of any other task. Otherwise, we insert an idle time until the release of the next future job with the earliest deadline. We show that CW-EDF is able to schedule any feasible harmonic and loose harmonic task set with period ratio greater than 3. In a loose harmonic task set, each period is an integer multiple of the smallest period [13].

The remainder of the paper is organized as follows; in Sect. II we describe our system model. In Sect. III we gather insightful examples to show the advantages and weaknesses of the existing non-preemptive algorithms. In Sect. IV the new necessary schedulability test is introduced. CW-EDF is presented in Sect. V and evaluated in Sect. VI. Finally the paper is concluded in Sect. VII.

II. System Model

We consider a uniprocessor system with a set of n independent, non-preemptive periodic tasks. The task set is denoted by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i, $1 \leq i \leq n$ is identified by $\tau_i : (C_i, T_i)$, where $C_i \in \mathbb{R}$ is the worst-case execution time (WCET) and $T_i \in \mathbb{R}$ is the period. Deadlines are implicit, i.e., they are equal to the period. We assume all tasks are released synchronously at time 0. Synchronous periodic task sets are a special case of *concrete periodic task sets* which can have an arbitrary, yet, known and positive release offset (according to [7]). A concrete periodic task set is *schedulable* with a scheduling algorithm A if using algorithm A none of the jobs generated by the tasks miss their deadlines in the time window from time 0 to ∞.

The system utilization is denoted by $U = \sum_{i=1}^{n} u_i$ where $u_i = C_i/T_i$ is the utilization of task τ_i. The hyperperiod is shown by H and is the least common multiple (LCM) of the periods. We assume that the tasks are indexed according to their period so that $T_1 \leq T_2 \leq \ldots \leq T_n$. The period ratio of task τ_i $(1 < i \leq n)$ is denoted by $k_i = T_i/T_{i-1}$, $k_i \in \mathbb{R}$ and it represents the ratio of two consecutive periods. We define the *period ratio of the task set* as $K^{min} = \min\{k_i\}_{i=2}^{n}$. The task set is harmonic if $\forall i, 1 < i \leq n, k_i \in \mathbb{N}$. In a harmonic task set, all tasks with smaller period will be released together with any release of a task with larger period. The task set is called *loose harmonic* [13] if $\forall i, 1 < i \leq n, T_i/T_1 \in \mathbb{N}$. In a loose harmonic task set, the releases of any task such as τ_i are synchronous with the releases of τ_1. We assume that there is only one task with the smallest period in the system. If in the original task set there are more than one task with the same period as T_1, we group them as one task with the execution time equal to the sum of execution times of the those tasks.

Besides, in order to make a consistent definition for the EDF algorithm, we assume that if two jobs have the same deadline, the one with the smaller index is scheduled first.

III. Motivations and Challenges

We start this section by revisiting the existing results on work-conserving and non-work-conserving scheduling algorithms to provide a better understanding about their advantages and weaknesses. Then we provide a summary of possible solutions as well as challenges in the way of designing new online non-work-conserving scheduling algorithms for concrete periodic tasks.

A. Work-Conserving Scheduling

In 1991, Jeffay et al. [7] have proven that *NP-EDF is optimal in the class of work-conserving non-preemptive scheduling algorithms for periodic task sets*. This work have been cited in more than 500 papers and reports. Due to the large influence of this work on the papers published in real-time systems, it is important to have a correct understanding of what has been claimed by that paper. Jeffay et al., have defined the *schedulable periodic task set* as [7]:

Definition 1. A periodic task set is schedulable if and only if the tasks can be scheduled for *any* set of release offsets.

According to Definition 1, if a task set is schedulable with a particular set of offsets whilst it is not schedulable by another set of offsets, that task set is not schedulable. Thus, the results of [7] become very pessimistic if they are applied to evaluate schedulability of a task set with known and fixed release offsets, i.e., a concrete periodic task set. This pessimism is strong enough to reduce the class of the hardness of the problem from NP-Hard to a problem which is solvable in pseudo-polynomial time using the necessary and sufficient schedulability test presented in [7]:

Theorem 1 (From [7]). *Task set τ with periodic tasks sorted in non-decreasing order of their periods is schedulable if $U \leq 1$ and $\forall i, 1 < i \leq n; \forall L, T_1 < L < T_i$:*

$$L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_j} \right\rfloor C_j \tag{1}$$

In order to clarify the consequences of Definition 1, we show that in contrast to the result of Jeffay on the optimality of NP-EDF in the class of work-conserving non-preemptive scheduling algorithms, there are task sets which are schedulable by non-preemptive fixed-priority (NP-FP) scheduling algorithm while they are not schedulable by NP-EDF (or NP-RM) if the release offsets are known and fixed.

Theorem 2. *NP-EDF is not optimal in the class of work-conserving non-preemptive scheduling algorithms for concrete periodic task sets.*

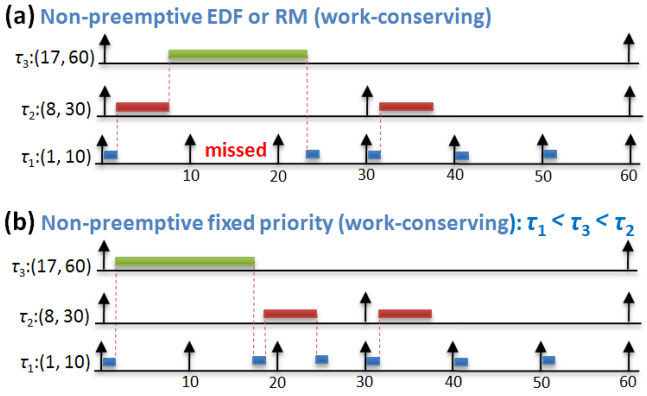

Figure 1. An example to show that NP-EDF is not optimal in the class of work-conserving algorithms if it is used in periodic task sets with known release offsets.

Proof: The proof is based on a counter example with 3 tasks: $\tau_1 : (1, 10)$, $\tau_2 : (8, 30)$, and $\tau_3 : (17, 60)$ all released at time 0. As shown in Fig. 1-(a), this task set is not schedulable by NP-EDF (or NP-RM) since the second job of τ_1 will miss its deadline due to the execution of τ_3 from 9 to 26. However, by assigning priority 3 to task τ_2 and priority 2 to task τ_3 we can schedule it without deadline miss, as shown in Fig. 1-(b). Note that even if the execution time of the tasks has variations at run-time, this task set remains schedulable under NP-FP. ■

Theorem 2 shows that in non-preemptive scheduling of concrete periodic tasks, NP-EDF does not dominate NP-FP in terms of schedulability. The next question is whether we can use Audsley's optimal priority assignment (OPA) algorithm [14] to obtain optimal priorities which guarantee the schedulability of NP-FP or not. Recently, Davis et al., [15] have used this algorithm to find such an optimal assignment for non-preemptive sporadic tasks as well as non-concrete tasks (which have unknown release offsets), however, prerequisites of Audsley's test are not satisfied in concrete periodic tasks because the priority ordering of lower priority tasks affects the response time of the higher priority tasks. For example, if we add $\tau_4 : (2, 120)$ to the example in Fig. 1-(a) and use OPA to assign its priority, we cannot ignore the priority ordering of the higher priority tasks because some priority orderings such as $\tau_1 < \tau_2 < \tau_3$ (where operator "<" shows the order of priorities from high to low),are unschedulable. While if priority ordering $\tau_1 < \tau_3 < \tau_2$ was used, the task set would become schedulable and hence, τ_4 could be assigned to the lowest priority. In other words, in the case of concrete periodic tasks, we probably need to iterate over all possible orderings of the tasks in order to use OPA algorithm. This in turn, increases the computational complexity of the algorithm to $O(n!)$.

In a recent work, Nasri et al., [16] have shown that in concrete periodic task sets the schedulability test in Theorem 1 becomes pessimistic, i.e., there are many NP-EDF-schedulable task sets which are rejected by the test. Here we extend their results and show that this test rejects any task set with $C_i > T_1 - C_1$ (for $1 < i \leq n$).

Theorem 3. *The schedulability test for NP-EDF in Theorem 1 rejects any task set with $C_i > T_1 - C_1$ (for $1 < i \leq n$).*

Proof: The proof is a direct result of Theorem 1. Assume τ_i is the task with $C_i = T_1 - C_1 + \Delta$ with $\Delta > 0$. In order to prove the claim, we use the same strategy used in [7] to prove the test itself: we show that the test cannot accept the task set because then there will be a particular release offset which causes a deadline miss for one of the tasks in the system. Since the test is designed to accept a task set only if it is schedulable for all possible release offsets, it must reject a task set with $0 < \Delta$. We assign the release offsets as follows: $R_i = 0 < \epsilon < \Delta$ and for any other task τ_j, $1 \leq j \leq n$, $j \neq i$, $R_j = \epsilon$. Since NP-EDF is work-conserving, it schedules τ_i from 0 to $T_1 - C_1 + \Delta$ because no other task is released in the system yet. Consequently, when τ_1 is released at time ϵ, it will be blocked until $\epsilon + T_1 - C_1 + \Delta$, which is larger than its deadline at $T_1 + \epsilon$. It means that τ_1 will miss its deadline, and hence, Theorem 1 cannot accept the task set. ■

Theorem 3 becomes interesting when we compare it with the necessary schedulability condition of concrete periodic tasks. This condition has been derived in the early works of [17] and [8] as

$$\forall i, 1 < i \leq n;\ C_i \leq 2(T_1 - C_1) \qquad (2)$$

By comparing (2) with the result of Theorem 3 we see that the maximum WCET of a task set which is acceptable by Theorem 1 is just half of what it could have been if the tasks have fixed release offsets. This limitation holds for many recent results on non-preemptive schedulability analysis of fixed-priority and dynamic priority scheduling algorithms in sporadic or non-concrete tasks, such as [18] and [15]. In other words, their results will be too pessimistic if they are used for periodic tasks with known release offsets. Unfortunately, This pessimism may not be easily surpassed as long as we consider work-conserving scheduling algorithms.

The natural difference between a work-conserving and non-work-conserving policy lies in the possibility of scheduling an idle interval; in the former approach, the processing resource must not remain idle as long as there is a pending job to execute. Although there can be examples, such as the one in Fig. 1, where a work-conserving algorithm is able to schedule a task with $C_i > T_1 - C_1$, it cannot be easily employed in the design of the schedulability tests because then we need to consider all combinations of the releases of the jobs of the tasks. For example, if in Fig. 1-(a), T_3 was 39 instead of 60, the task set would be unschedulable even with the NP-FP algorithm. The following theorem shows that there are feasible task sets which are not schedulable by *any* work-conserving scheduling algorithm.

Theorem 4. *There exists feasible non-preemptive task set τ which cannot be scheduled by any work-conserving scheduling algorithm.*

Proof: The proof is based on a counter example with

3 tasks $\tau_1 : (1, 5)$, $\tau_2 : (1, 10)$, and $\tau_3 : (8, 20)$. Regardless of the execution order of the first job of τ_1 and τ_2, a work-conserving algorithm schedules τ_3 at 2, which then, causes a deadline miss for the next τ_1. If τ_3 is scheduled before τ_2, the first job of τ_2 will miss its deadline. Since a work-conserving algorithm cannot leave the processor idle, there is no other combination of priorities (job ordering) that guarantees the schedulability. However, this task set is schedulable if RM priorities are used and an idle interval is inserted from 2 to 5. ■

As a conclusion from Theorems 2, 3, and 4, we see that there is a natural limitation in the schedulability of work-conserving scheduling algorithms for non-preemptive task sets with fixed release offsets.

From another perspective, there have been efforts to obtain the speedup factor which guarantees the schedulability of NP-EDF for sporadic tasks [19], [18], [15]. In [19], the speedup is upper bounded by $4C^{max}/D^{min}$ where C^{max} is the maximum execution time and D^{min} is the smallest deadline. Nasri et al., [12] have shown that in special cases of harmonic task sets with synchronous release this bound is 8 (in other cases it can be larger). However, practically speaking, it is hard (or sometimes impossible) for a system designer to provide such large speedup factor. If the tasks are messages which must be sent over a network such as CAN network, the transmission speed and the length of messages are fixed. Moreover, increasing the speed of the processing unit (e.g., CPU) comes with the increase in the size, cost, and energy consumption of the system, which are usually limited in many embedded systems. Besides, as stated before, work-conserving scheduling algorithms have natural limitation in scheduling many feasible task sets. Hence, by moving towards non-work-conserving algorithms, we increase the schedulablity without any need to augment the resources (e.g., speeding up the CPU). Even for the systems which can have speedup, there will be more chance to have smaller speedup factor with non-work-conserving algorithms than work-conserving ones.

B. Non-Work-Conserving Scheduling

Recently, an online scheduling algorithm called Precautoius-RM has been introduced by Nasri et al. [12], [13]. Using this algorithm, they have shown that (2) is a necessary and sufficient schedulability condition for harmonic or loose harmonic task sets with $\forall i; 1 < i \leq n; k_i \geq 3$ where k_i is the period ratio of two consecutive tasks. This result holds even for the cases where $\forall i; 1 < i \leq n; k_i = 2$. From these results together with Theorem 4 we conclude that: 1) the hardness of the non-preemptive scheduling problem for concrete periodic tasks depends on the period ratio of the tasks, 2) non-work-conserving scheduling policies bring new potentials for scheduling non-preemptive tasks, 3) as shown by Theorem 2, in non-preemptive scheduling, EDF policy is not necessarily the best choice since there are task sets which are schedulable by fixed-priority scheduling algorithm.

Same as the existing scheduling policies such as RM and EDF, we are able to have non-work-conserving equivalent policies, where the job priority is assigned by the underlying job scheduling algorithm, e.g., FP, RM, EDF, etc., while the the *idle-time insertion policy* (IIP) decides whether an idle interval must be scheduled or not. In the latter case, it also determines the length of that interval.

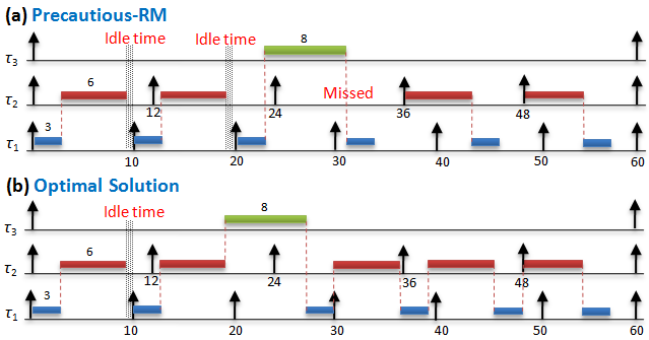

Figure 2. An example with 3 tasks $\tau_1 : (3, 10)$, $\tau_2 : (6, 12)$, and $\tau_3 : (8, 60)$. This example shows that the necessary condition of schedulability in (2) may be very loose if period ratio is not larger than 3. This task set is feasible, yet it is not schedulable by any work-conserving scheduling algorithm or Precautious-RM.

In Precautoius-RM (shown in Algorithm 1), IIP is described as a condition which is verifiable in $O(1)$ as follows. Assume τ_i is the current high priority task with a pending job. It can be scheduled if one of the two following conditions hold: a) the previously executed task was τ_1, and τ_i will not cause a deadline miss for the next job of τ_1, or b) τ_i will be finished before the next release of τ_1. Otherwise, the algorithm schedules an idle interval until the next release of τ_1. This idle interval plays the synchronization role according to the releases of τ_1. After that, τ_i may have a chance to fit into the slack of two consecutive jobs of τ_1, which is the longest interval of time where a non-preemptive task can be scheduled.

Algorithm 1: Precautious-RM

Input: t: the current time

$\tau_i \leftarrow$ the highest priority task with a pending job;
$r_1^{next} \leftarrow$ release time of the next job of τ_1 after t;
if *($(t + C_i \leq r_1^{next})$ **or** $(t + C_i \leq r_1^{next} + T_1 - C_1$ **and** τ_1 is the latest executed task))* **then**
  Schedule τ_i;
else
  Schedule an idle interval from t to r_1^{next};
end

The IIP in Precautious-RM makes it an optimal algorithm for concrete periodic task sets with special period ratios (e.g., for loose harmonic tasks with $k_i \geq 3$), however, due of the fact that the idle-times are always inserted until the next release of τ_1, they might waste slack of some of the low priority tasks and cause a deadline miss in cases where period ratio is smaller than 3. The example in Fig. 2-(a) shows a task set which is not schedulable by either Precautious-RM or any work-conserving scheduling solution, yet it is *feasible* as shown in Fig. 2-(b). This example shows the importance of having an efficient IIP.

C. Challenges

As it has been done in Precautious-RM, it is possible to combine different job scheduling policies with idle-time insertion, and hence, create new non-work-conserving policies that increase the schedulability of non-preemptive task sets. However, we must find an efficient IIP to determine when and for how long an idle interval must be scheduled. This problem has some connections with the problem of determining the maximum tolerable blocking time by a task. If we could know the maximum amount of time for which we could safely block a high priority job, then at time t when we decide to schedule current job of τ_j, we could look into the high priority jobs which will be released from t to $t + C_j$ and check whether they can tolerate such a blocking or not.

In the context of preemptive work-conserving scheduling, the maximum tolerable blocking has been derived for many algorithms. For example, Yao et al., [20] have formulated it for fixed-priority preemptive scheduling as

$$\beta_i = \max_{C_i < t \leq T_i} \{t - rbf_i(t)\} \tag{3}$$

where $rbf_i(t)$ is the request bound function (RBF) and shows the upper bound on the execution demand in a time window of length t where higher priority tasks than τ_i execute:

$$rbf_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{4}$$

However, unlike preemptive task sets, idle intervals might be unavoidable to guarantee the feasibility of a non-preemptive task set. Consequently, the β_i calculated in (3) loses its meaning because then the real tolerable blocking can be much smaller than β_i. For example, using (3) for the task set in Fig. 2-(b), β_3 will be 4 while due to the unavoidable idle interval from 9 to 10, the real blocking tolerance is 3. Consequently, in order to find similar analysis based on the notion of demand and supply for non-preemptive task sets, we need to account for idle intervals, either in the demand or in the supply functions. For example, we can have non-linear supply functions where they do not increase during the idle intervals. Then, the schedulability could be verified if the existing demand always stays below the available supply for any interval of time. However, without having an optimal policy for idle-time insertion, we cannot say where and when the idle intervals must appear in the supply bound function. In other words, it will become a cyclic problem which is hard to solve.

Another approach to tackle the problem of defining an IIP is through assuring that the decisions taken at time t, will not cause a deadline miss for a certain horizon of time or certain tasks in the future. A very simple version of this technique is used in Precautious-RM where it only looks after the next job of τ_1 in the future. Assume that at time t, we want to decide whether to schedule the current job of task τ_i or not, and if not, we schedule an idle interval from t to $t + L$. In other words, in order to design a safe online IIP, we just need to quantify the upper bound on the number of jobs of different tasks which we must consider into account before we decide to whether schedule τ_i or an idle interval. According to the results of Precautious-RM, we know that if the task set is harmonic (with period ratio two or greater than two), or loose harmonic (with period ratio greater than or equal to 3) the aforementioned upper bound on the number of jobs is 1, and that job is the next instance of τ_1.

Despite the fact that IIP of Precautious-RM is efficient in many task sets, it is not enough for all of them, particularly when the period ratio is smaller than 3 where the hardness of the problem lies. In that case, even the existing necessary schedulability condition defined in (2) becomes loose. For example, in the task set in Fig. 2, the maximum permissible C_3 is not 14 which is two times slack of τ_1, but instead, it is 8. In order to provide a solution for the cases where period ratio is smaller than 3, in this paper we tighten the necessary schedulability condition. Based on the idea of our necessary test, we design an IIP to be used by EDF. In this IIP, instead of the next job of τ_1, we watch over n future jobs (one job from each task). This scheduling algorithm will be presented in Sect. V.

IV. Tightening the Necessary Condition of Non-Preemptive Scheduling

In this section, we find a necessary condition which is tighter than (2) for the schedulability of non-preemptive tasks. As shown in Fig. 2-(b), if the period ratio is smaller than 3, the execution time of tasks such as τ_j, $2 \leq j < i$ may affect the maximum feasible execution time of τ_i. This value will be denoted by C_i^{max}, and it shows the upper bound of the largest value which can be assigned to C_i without jeopardizing the schedulability of any other task in a feasible schedule.

As shown in the example in Fig. 2-(b), C_i^{max} may not necessarily happen between two consecutive releases of τ_1. In this example, τ_3 has started its execution while the second job of τ_1 which is released at 10 is already finished before the second job of τ_2 released at 12. Consequently, the second job of τ_1 will not interfere with the non-preemptive interval $[19, 27]$ which is used to schedule τ_3. Thus, a scenario such as the one shown in Fig. 3 will not necessarily find the largest non-preemptive interval.

We first define *interfering jobs* of a task such as τ_i within interval t as follows

Definition 2. Interfering jobs of a periodic task such as τ_i in the interval $[t_1, t_2]$ are the jobs that their release time and deadline is in the given interval.

The minimum interfering workload generated by the jobs of τ_i in any interval of time of length t can be obtained as

$$I_i(t) = \max\left\{0, \left(\left\lfloor \frac{t}{T_i} \right\rfloor - 1\right) C_i\right\} \tag{5}$$

because in every interval of length t there will be at least $\lfloor t/T_i \rfloor$ releases of τ_i. Among these $\lfloor t/T_i \rfloor$ jobs, at least $\lfloor t/T_i \rfloor - 1$ must have their release and deadline within the interval of length t for any possible release

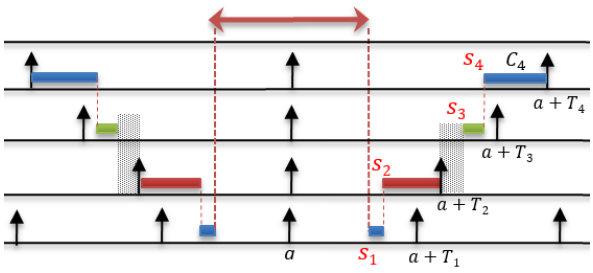

Figure 3. An example to show that calculating the maximum slack of a set of jobs which are released together will not necessarily lead to a necessary condition. If this scenario is applied on the task set in Fig. 2-(b), C_3^{max} will be 6 while we see that the real value is 8.

offset. Consequently, (5) derives the lower bound on the interfering workload.

For example, in Fig 2-(b), in any interval of length 24, τ_1 has at least $\lfloor 24/10 \rfloor - 1 = 1$ interfering job which has its release and deadline within the 24 units of time. It is worth noting that if $t < T_i$, the right-hand-side of (5) becomes 0, which means that tasks with larger periods than the given interval do not contribute to the minimum interfering workload.

Our idea for the necessary test is to calculate the lower bound on the workload of tasks with smaller periods which *must* be executed within every interval of time of length t. The following theorem presents this test.

Theorem 5. *A task set τ defined in Sect. II is not schedulable by any non-preemptive scheduling algorithm if for at least one task τ_i $(1 < i \leq n)$, $C_i > C_i^{max}$ where for $1 < i \leq n$,*

$$C_i^{max} = \min \{\theta_j\}_{j=1}^{i-1} \tag{6}$$

$$\theta_j = 2(T_j - C_j) - \sum_{p=1}^{j-1} I_p(2T_j) \tag{7}$$

Proof: The proof for i=2 is trivial as it is identical with the necessary test in (2). Since preemption is not allowed, any feasible non-preemptive task τ_i will eventually be scheduled in the slack available within two consecutive jobs of τ_j $(j < i)$. This slack is denoted by θ_j. Hence, the minimum value among all of the available slack θ_j gives C_i^{max} as denoted by (6). In order to find θ_j, we sum up the minimum interfering workload generated by tasks with smaller period than τ_j within an interval of time of length $2T_j$. Task τ_j will have exactly 2 interfering jobs in this interval because the interval starts from the release of τ_j. For this reason we have term $2(T_j - C_j)$ in (7). Then according to (5), the minimum interference from jobs of tasks with smaller period than τ_j is calculated by summing up their individual interference which proves (7).

Since (6) provides an upper bound on the available slack to schedule task τ_i without jeopardizing the tasks which *must* be scheduled in that interval of time, if $C_i > C_i^{max}$, at least one of those interfering tasks will miss its deadline. ■

If we apply (6) on the example in Fig. 2, $C_2^{max} = 2(10 - 3) = 14$ and $C_3^{max} = 2(12 - 6) - 3(\lfloor 24/10 \rfloor - 1) = 9$ which is tighter than what we get from (2). It is worth noting that our test in Theorem 5 is a linear-time test because both (7) and (6) can be calculated in $O(n)$.

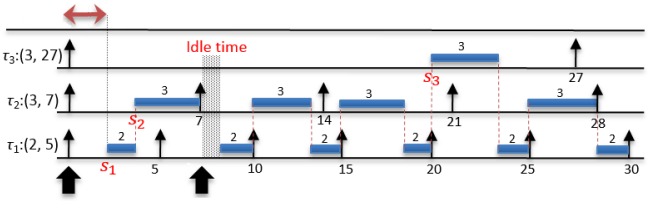

Figure 4. An example to show that finding the latest feasible start time of a set of jobs in a non-work-conserving solution may involve considering a large number of jobs and scheduling idle times.

One way to tighten the necessary test in Theorem 5 might be through calculating the amount of unavoidable idle times in a feasible schedule. However, as shown in Fig. 2-(b), for doing that we should solve the non-preemptive scheduling problem in the first place in order to know where to insert those idle times. Yet, for an online decision making about scheduling a task or an idle interval, we might want to take a look into the future to know whether scheduling τ_i will cause a deadline miss for any other task or not. Hence, we are interested to find a the latest start time of a set of jobs at run time. It can be done through specifying the maximum length (or the maximum number of jobs) of the longest possible chained interference which can happen in a task set. One example is shown in Fig. 4. As shown here, at time 0, the latest start time of the first job of τ_1 only depends on the first job of τ_2 while at time 7, a long chain of jobs must be considered in order to find the latest start time of τ_1. The maximum number of jobs in a chained interference depends on the properties of the tasks. For example, from the results of Precautious-RM [12], [13] we know that if the task set is loose harmonic with period ratio greater than 3, then the maximum number of jobs which may affect the optimality of the decisions is 1, and that job is the next instance of τ_1. Having such insightful information about a task set will help us to create task-set-aware IIP policies, i.e., an IIP that employs design-time information of the task set at run-time in order to guarantee schedulability.

V. Non-Work-Conserving EDF Scheduling Based on Critical Time Windows

In this section, we introduce a non-work-conserving version of EDF which watches over a certain set of jobs in the future. We define a *critical job* as a job whose next release has the earliest deadline among the jobs which will be released in future. We call the deadline of the critical job as *critical time window*, accordingly, we name our scheduling algorithm CW-EDF. This algorithm is online and is activated in one of the following events: a) finishing of a job, b) finishing of a scheduled idle interval, and c) a job release when no other task is executing or an idle interval is scheduled.

Pseudo-code of CW-EDF is presented in Algorithm 2. Fig. 5 illustrates the steps of the algorithm and the notations. Assume that CW-EDF is activated at time t and the

current job of τ_i, denoted by J_i, has the earliest deadline among other jobs in the system. We denote the critical time window by $\omega_i(t)$. In the first step, the algorithm finds the critical job by calculating the deadline of the next job of each task τ_k, $1 \leq k \leq n$:

$$D_k^{next}(t) = \left(\left\lceil \frac{t}{T_k} \right\rceil + 1\right) T_k \tag{8}$$

Consequently, the critical time window will be

$$\omega_i(t) = \min\{D_k^{next}(t)\}_{1 \leq k \leq n, k \neq i} \tag{9}$$

Since EDF is a job-level fixed priority scheduling algorithm, none of the pending jobs in the system will have a better chance to be scheduled as long as J_i (which has the earliest deadline) is still in the system. Due to the fact that we insert idle intervals to protect the schedulability of the tasks which will be released soon and may have short amount of blocking tolerance, if the critical job is not going to be scheduled, other low priority jobs will have to wait more. That is why, in the critical window, we do not consider the effect of their current pending jobs, instead, we only look into the next release of each task.

After finding $\omega_i(t)$, with the same analogy used in our new necessary schedulability condition, we calculate the possible interference of the future jobs. Same as Sect. IV, we only consider one job from each task. First we created a sorted list of indices of the tasks that do not have a pending job at time t in the system. We denote this list by I and it has m items. Then we sort this list based on the deadlines in ascending order from the earliest to the latest deadline. Starting from the last item in the list, i.e., J_{I_m} we calculate $L_m(t) = D_m^{next}(t) - C_m$, where $L_m(t)$ is the latest start time of J_{I_m}. Then we use the following equation to obtain the latest start time of the other jobs until we reach to the critical job which is J_{I_1}. This process has been shown in Fig. 5.

$$L_p(t) = \min\{D_p^{next}(t), L_{p+1}(t)\} - C_p \tag{10}$$

In our approach, if the resulting $L_1(t)$ is smaller than the release of the critical job itself, a deadline miss is unavoidable. However, if $t + C_i < L_1(t)$, we know that by scheduling J_i we will not force a deadline miss to the critical job.

Algorithm 2: Critical Time Window EDF (CW-EDF)

Input: t: the current time

```
Set J_i as a job with the earliest deadline;
Obtain L_1(t) from (10);
if (t + C_i ≤ L_1(t)) then
    Schedule J_i;
else
    Schedule an idle interval from t until the release
      of the critical job;
end
```

Algorithm 2 has $O(nlog(n))$ computational complexity since we create a sorted list of the future deadlines of each task which is not currently in the system (does not have a pending job in the system).

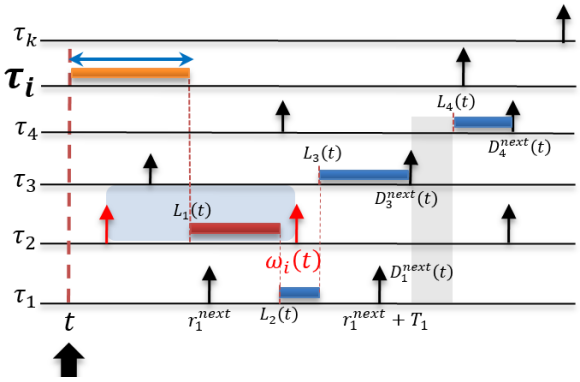

Figure 5. A symbolic example to illustrate the notations used in the algorithm and the rationale of the decision.

A. On the Schedulability of CW-EDF

In this section we discuss the schedulability of CW-EDF, mostly for harmonic and loose harmonic tasks. The reason is to show that CW-EDF keeps the schedulability in the previously known solvable situations. In other words, our goal here is mostly to show that CW-EDF can schedule task sets which were schedulable by Precautious-RM. First we start by the fact that in harmonic and loose harmonic task sets, the job which creates the critical time window is always the next job of τ_1.

Lemma 1. *In harmonic and loose harmonic task sets with synchronous release, the next job of τ_1 will always be the job with deadline at $\omega_i(t)$ (obtained from (9)).*

Proof: If at time t, J_i $(1 < i \leq n)$ is the highest priority job, it means that the last instance of τ_1 is already scheduled and is no longer the job with the earliest deadline. Since in harmonic or loose harmonic tasks, all releases are synchronous with the releases of τ_1, the next release of any other job in the system cannot be between t and r_1^{next} where r_1^{next} is the next release of τ_1. Moreover, since τ_1 has the smallest period, even if there are tasks which are released synchronously at r_1^{next}, value of $\omega_i(t)$ is still equal to the deadline of τ_1 at $r_1^{next} + T_1$ since it will be the earliest deadline. ■

Next we show that if the minimum period ratio of the tasks in a harmonic or loose harmonic task set is larger than or equal to 3, $L_1(t)$ from (10) will never be smaller than the time remained from the slack of the current τ_1 plus the slack of the next job of τ_1.

Lemma 2. *In harmonic and loose harmonic task sets with synchronous release and period ratio $k_i \geq 3$ (for $1 < i \leq n$), the remaining time until $L_1(t)$ (from (10)) will never be smaller than $r_1^{next} - t + T_1 - C_1$ where t is the time that the algorithm is activated and r_1^{next} is the release time of the next job of τ_1.*

Proof: If $k_i \geq 3$, we will have $D_m^{next}(t) \geq 3T_{m-1}$. Besides, according to (2), $C_i \leq 2(T_1 - C_1)$. Consequently, in (10) starting from $L_m(t) = D_m^{next}(t) - C_m$, for each value of $L_p(t)$, the minimum between $D_p^{next}(t)$ and $L_{p+1}(t)$ will be $D_p^{next}(t)$. As a result, $L_1(t) = D_1^{next}(t) - C_1$. However, according to Lemma 1, the critical job is

the next job of τ_1, and hence, $D_1^{next}(t) = r_1^{next} + T_1$ and hence, $L_1(t) = r_1^{next} + T_1 - C_1$. Therefore, the remaining time until the latest start time of the critical job will be $r_1^{next} + T_1 - C_1 - t$. ■

Now we present the following lemma which simplifies the scheduling conditions upon which Precautious-RM is built. (Note that since the proof of the following lemma comes from two already published proofs from [12], [13], we just explain them briefly).

Lemma 3. *In harmonic and loose harmonic task sets with synchronous release and period ratio $k_j \geq 3$ (for $1 < j \leq n$), an online scheduling algorithm which follows RM priorities and will schedule an idle interval from t until r_1^{next} if $t+C_i > r_1^{next}+T_1-C_1$, guarantees the feasibility of the schedule. Here t is the time that the algorithm is activated, r_1^{next} is the release time of the next job of τ_1 after t, and τ_i is the high priority task with a pending job.*

Proof: This proof directly follows the proof of schedulability of Precautious-RM in Theorem 3 from [12]. Note that those proofs are based on the worst-case scenario where tasks have execution time equal to $C_j = 2(T_1 - C_1)$, which means that the other two conditions in Line 3 of Algorithm 1 (Precautious-RM) will never become true and the only condition which can be used to guarantee the schedulability is $t + C_i > r_1^{next} + T_1 - C_1$. The second part of the proof for loose harmonic tasks comes from Theorem 10 in [13]. Same as the previous one, this proof is based on the worst-case scenario where $C_j = 2(T_1 - C_1)$. It means that the only required condition for the schedulability is to verify $t + C_i > r_1^{next} + T_1 - C_1$. If it is not satisfied, then an idle interval must be inserted until the next release of τ_1. ■

Putting Lemma 1, 2, and 3 together we show that CW-EDF is able to schedule any harmonic and loose harmonic task set with $k_i \geq 3$ which have the necessary condition of schedulability, i.e., Condition (2).

Theorem 6. *CW-EDF is able to guarantee the schedulability of any harmonic and loose harmonic task set with synchronous release and period ratio $k_i \geq 3$ (for $1 < i \leq n$) as long as the necessary condition of schedulability in* (2) *holds.*

Proof: The proof is based on the fact that according to Lemma 2, the decision made by CW-EDF is just based on the next job of τ_1, therefore, its scheduling condition in Line 3 becomes identical to the scheduling condition required by Lemma 3 to guarantee the schedulability. Besides, since in the harmonic and loose harmonic task sets, the earliest release is always synchronous with the releases of τ_1, the idle times inserted by CW-EDF are identical with the idle times suggested by Lemma 3, which means that CW-EDF is able to schedule those task sets. ■

Although we have shown that CW-EDF can schedule many cases of harmonic and loose harmonic task sets, its strength is in its capability to schedule periodic task sets with period ratio smaller than 3. As we will show in Sect. VI, CW-EDF is very efficient for such task sets. It emphasizes on the importance of developing a more general schedulability test for CW-EDF as a part of our future works.

VI. Experimental Results

In this section we evaluate the performance of our proposed scheduling algorithm and proposed necessary test in two separate sets of experiments.

A. Evaluating Schedulability Ratio

In the first set of experiments, we compare the performance of CW-EDF with NP-EDF, NP-RM, Precautious-RM [21], [13], and Gr-EDF [11]. Our performance measure is the schedulability ratio which is the ratio of schedulable tasks to the total number of generated task sets. If a task set has a deadline miss in the hyperperiod, it is considered to not be schedulable. Each simulation experiment averages 500 simulation runs (each including 8 randomly generated tasks). Due to the space limitation, we did not report the results for larger task sets, however, the performance of the algorithms is almost the same as shown in the second set of experiments.

In the first experiment, we have considered the effect of minimum period ratio, i.e., K^{min} (defined in Sect. II) in periodic tasks. In order to generate random task sets, we first choose a random value for T_1 and u_1 from ranges $[1, 10]$ and $[0.01, 0.99]$ with uniform distribution, respectively, and then, we have assign $C_1 = u_1 T_1$. Afterwards, for every task τ_i $(1 < i \leq n)$ we have selected random $k_i \in [K^{min}, 4]$ with uniform distribution, where $k_i \in \mathbb{R}$. Then we assigned each period as $T_i = k_i T_{i-1}$, thus, the resulting periods are not necessarily harmonic or loose harmonic. The execution times have been selected randomly with the uniform distribution from $[0.01, 2(T_1 - c_1)]$. Because of the choice of periods, hyperperiod might become very large. In this experiment, we ignore the task sets with more than 100,000 jobs per hyperperiod. Besides, we have performed the experiments on task sets that respect our feasibility condition given in Theorem 5.

As shown in Fig. 6, the schedulability ratio of CW-EDF is greater than Precautious-RM which is one of the efficient state of the art methods. Particularly, when the period ratio of the tasks is small, Precautious-RM does not have a good performance as it inserts many idle times which waste the chance of other tasks to be scheduled. The counter example in Fig. 2-(a) shows one of these task sets. Moreover, other scheduling algorithms have considerably small schedulability ratio because they do not insert any idle time even when it is necessary. The significant gap between the performance of work-conserving and non-work-conserving scheduling algorithms can be seen in this figure. While the average schedulability ratio of NP-EDF, NP-RM, and Gr-EDF is 15%, it is 82% for Precautious-RM and 90% for CW-EDF, which is considerably greater than the work-conserving solutions.

In the next experiment, we have considered the effect of maximum period ratio, i.e., K^{max}, in periodic and in loose harmonic task sets. The task generation process here is the

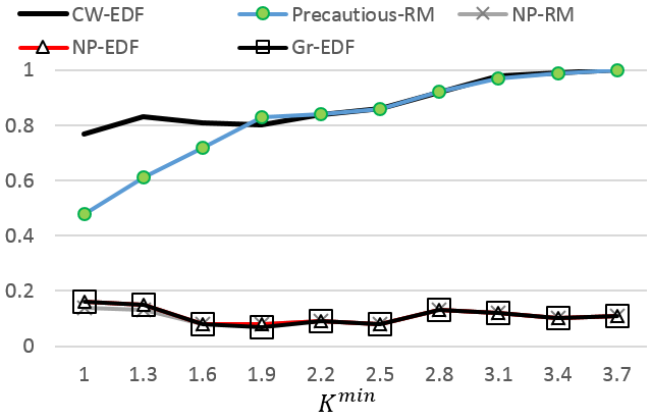

Figure 6. Schedulability ratio as a function of K^{min} in periodic task sets. In the diagram, NP-RM, NP-EDF, and Gr-EDF have overlapped.

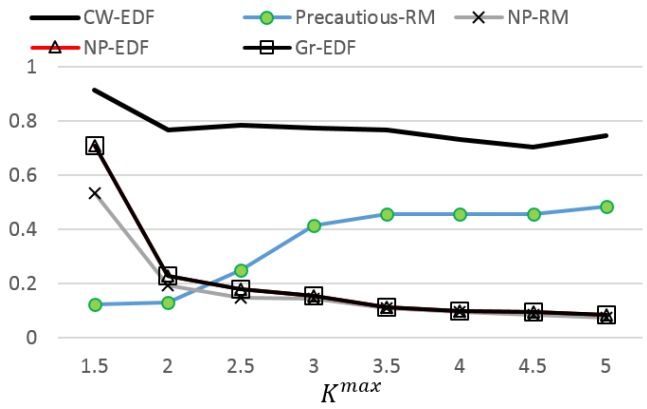

Figure 7. Schedulability ratio as a function of K^{max} in periodic task sets. In the diagram, NP-EDF and Gr-EDF have overlaped.

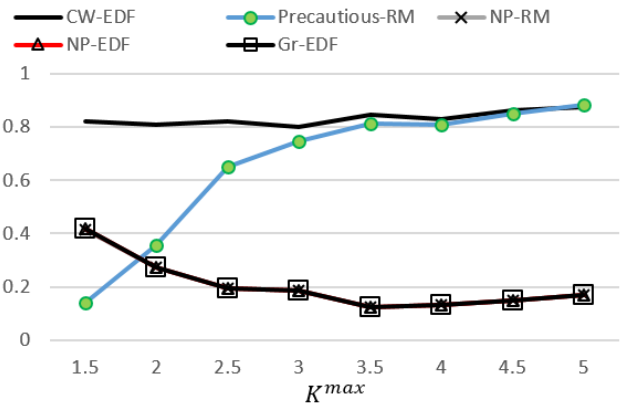

Figure 8. Schedulability ratio as a function of K^{max} in loose harmonic task sets. In the diagram, NP-EDF and Gr-EDF have overlaped.

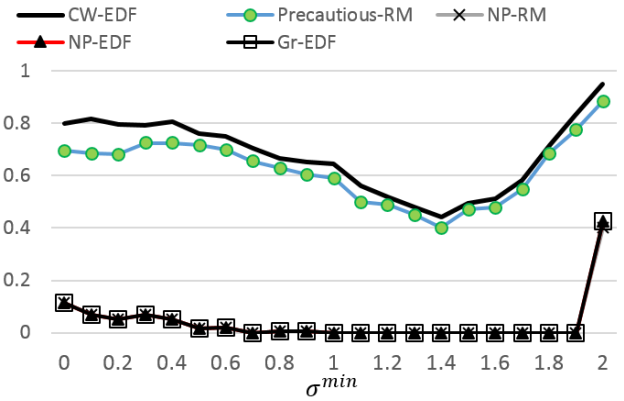

Figure 9. Schedulability ratio as a function of σ^{min}.

same as the previous experiment. The only difference is when we assign k_i, which is now a random value with uniform distribution from $[1, K^{max}]$. For loose harmonic tasks, after obtaining $k_i T_{i-1}$, we round it up to the next integer multiple of T_1 as $T_i = \lceil (k_i T_{i-1})/T_1 \rceil T_1$. The results of periodic tasks have been shown in Fig. 7 and the results of loose harmonic tasks have been shown in Fig. 8. Since in these task sets, the lower bound on the period ratio is 1, we will have more tasks with $k_i < 3$. As shown in Fig. 7, in periodic tasks, work-conserving algorithms have better schedulability than Precautious-RM when the maximum period ratio is smaller than 2.5. It is due to the fact that in such situations, inserting idle times must be done very carefully, otherwise, tasks will lose their slack due to the idle times, and then, cannot finish before their deadline. However, even though NP-EDF is able to schedule almost 63% of the task sets, there are still many more task sets which can be scheduled by adding idle times in a more careful way. In this figure, CW-EDF shows its significant potentials to schedule many more non-preemptive tasks that what could have been scheduled so far. The average schedulability ratio of CW-EDF is 78%, while Precautious-RM is 36%, and NP-EDF is 19%. By comparing Fig. 7 and 8 we see that Precautious-RM has much higher performance in loose harmonic tasks in comparison with periodic tasks because in loose harmonic task sets the low priority tasks will not release in the middle of a scheduled idle interval since the end of these intervals is synchronous with the releases of τ_1.

Next we evaluate the effect of σ^{min} on the schedulability. We consider $k_i \in [1, 4]$, however, while we generate C_i we force it to be selected as a random value from the following interval: $[\sigma^{min} \times 2(T_1 - C_1), 2(T_1 - C_1)]$. Doing so, we limit the lower bound of C_i with respect to $2(T_1 - C_1)$ by factor σ^{min}. The larger the σ^{min}, the larger the execution time of the tasks.

Fig. 9 shows the results of our experiment. When σ^{min} is larger than 1, the schedulability ratio of the work-conserving scheduling algorithms reduces significantly. When σ^{min} is between 1.2 and 1.8, tasks have execution times larger than the slack of τ_1, and hence, when they start, they push the execution of the next instance of τ_1 (which will be released during their execution) to a later time. As a result, when the next job of τ_1 is executed, the available slack of that instance is smaller than $T_1 - C_1$ (since a part of it was used by other tasks). If the next high priority task cannot fit into the remaining slack of τ_1 plus the slack of the next τ_1 (the third), then the algorithm needs to insert an idle time, otherwise, the third instance of τ_1 will miss its deadline. As a result, the amount of idle times which must be inserted increases. At the same

time, the execution time has already been increased due to σ^{min}, which in turn, increases the utilization of the task set. Having all these facts together we see that the performance of CW-EDF and Precautious-RM dropped by almost 20% where σ^{min} is large. Moreover, if $\sigma^{min} = 2$, all tasks will have exactly the same execution time, which is equal to $2(T_1 - C_1)$. This situation have helped NP-EDF and other work-conserving algorithms to have better schedulability than before.

B. Evaluating the Necessary Condition

In this set of experiments, we compare our necessary schedulability test with the one of Cai et al., [8] presented in (2). The performance measure is the acceptance ratio. Due to the NP-Hardness of the problem, we cannot compare the necessary tests with the real feasible task sets. Moreover, we only generate task sets which satisfy (2). We have considered the effect of σ^{min}, n, K^{min}, and K^{max} on the necessary conditions in periodic task sets. In the experiment on σ^{min}, periods have been selected randomly from $[10, 500]$, n=10, and $C_i \in [\sigma^{min} \times 2(T_1 - C_1), 2(T_1 - C_1)]$. The experiment on n is the same with the exception that $sigma^{min} = 1$. The experiments on K^{min} and K^{max} are the same as Sect. VI-A for $n = 10$.

As shown in Fig. 10-(a), when σ^{min} increases the acceptance ratio decreases because large σ^{min} affects the available slack of the tasks through increasing $I_i(t)$ in (5). It is worth noting that some of the tasks which are generated with our random task generation method may have $U > 1$, which is the reason that they are rejected by Cai's test too. According to Fig. 10-(b), $n > 5$ has nearly no effect in the performance of the test.

In Fig. 10-(c), when K^{min} is larger than 2.2, our necessary test almost accepts all task sets. This result is consistent with what we see in Fig. 6 because when K^{min} increases, the schedulability increases as well. Fig. 10-(d) shows that K^{max} has an important effect on the schedulability. In this diagram, the gap between two necessary tests reaches to 42% which shows the tightness of our new test.

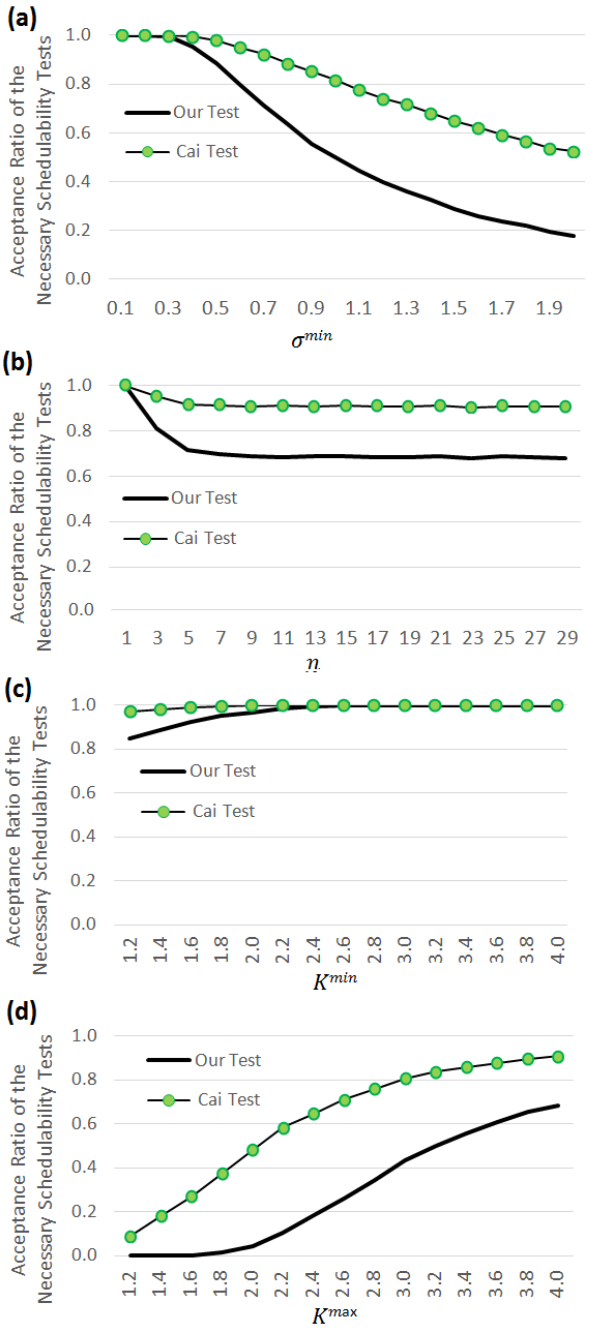

Figure 10. Acceptance ratio of the necessary condition by Cai [8] and our Theorem 5 based on σ^{min}, n, K^{min} and K^{max} in periodic tasks. Note we have generated random task sets which are consistent with (2). However, some of them may have $U > 1$, which is the reason for being rejected by Cai's test.

VII. Summary and Conclusion

In this paper we have discussed the problem of non-preemptive scheduling of periodic tasks. We show that the existing work-conserving scheduling algorithms and their schedulability tests are not able to efficiently cover this type of task sets. We have shown that despite the hardness of the problem, it is possible to have efficient non-work-conserving scheduling solutions. Then we discussed the weaknesses of the existing non-work-conserving scheduling algorithms and described the challenges in the design of an idle-time insertion policy which can be used together with the scheduling policies and make them efficient for non-preemptive tasks. As an example, we have developed an idle-time insertion policy for EDF.

Through the experimental results we have shown that our proposed scheduling algorithm is able to schedule 80% of the randomly generated periodic tasks while the existing work-conserving solutions schedule 15% and the existing non-work-conserving solution schedules 40% of those task sets. As a future work, we extend the schedulability test of our algorithm to cover periodic tasks. We will also provide extensions of the algorithm and schedulability test for task sets with release jitter.

Acknowledgements

We would like to thank Rob Davis, Geoffrey Nelissen, and our anonymous reviewers for their helpful comments.

REFERENCES

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

[2] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[3] M. Grenier and N. Navet, "Fine-tuning MAC-level protocols for optimized real-time QoS," *IEEE Transaction on Industrial Informatics*, vol. 4, no. 1, pp. 6–15, 2008.

[4] T. P. Baker, "Stack-based Scheduling for Realtime Processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, 1991.

[5] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems: A survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, 2013.

[6] C.-G. Lee, J. Hahn, Y.-M. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 700–713, 1998.

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

[8] Y. Cai and M. C. Kong, "Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems," *Algorithmica*, vol. 15, no. 6, pp. 572–599, 1996.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[10] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uni-processor scheduling," INRIA Research Report nRR2966, Tech. Rep., 1996.

[11] W. Li, K. Kavi, and R. Akl, "A non-preemptive scheduling algorithm for soft real-time systems," *Computers and Electrical Engineering*, vol. 33, no. 1, pp. 12–29, 2007.

[12] M. Nasri and M. Kargahi, "Precautious-RM: a predictable non-preemptive scheduling algorithm for harmonic tasks," *Real-Time Systems*, vol. 50, no. 4, pp. 548–584, 2014.

[13] M. Nasri and G. Fohler, "Non-Work-Conserving Scheduling of Non-Preemptive Hard Real-Time Tasks Based on Fixed Priorities," in *International Conference on Real-Time Networks and Systems (RTNS)*. ACM, 2015, pp. 309–318.

[14] N. C. Audsley, "On Priority Asignment in Fixed Priority Scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39–44, 2001.

[15] R. I. Davis, A. Burns, S. Baruah, T. Rothvoß, L. George, and O. Gettings, "Exact comparison of fixed priority and EDF scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms," *Real-Time Systems*, vol. 51, no. 5, pp. 566–601, 2015.

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

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

[18] G. von der Bruggen, J.-J. Chen, and W.-H. Huang, "Schedulability and optimization analysis for non-preemptive static priority scheduling based on task utilization and blocking factors," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 90–101.

[19] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Quantifying the sub-optimality of non-preemptive real-time scheduling," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2013, pp. 113–122.

[20] G. Yao, G. Buttazzo, and M. Bertogna, "Feasibility analysis under fixed priority scheduling with limited preemptions," *Real-Time Systems*, vol. 47, no. 3, pp. 198–223, 2011.

[21] M. Nasri, G. Fohler, and M. Kargahi, "A Framework to Construct Customized Harmonic Periods for Real-Time Systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 211–220.