New Schedulability Test Conditions for Non-preemptive Scheduling on Multiprocessor Platforms

Technical Report May 2008

Nan Guan[1], Wang Yi[2], Zonghua Gu[3] and Ge Yu[1]

[1] Northeastern University, Shenyang, China
[2] Uppsala University, Uppsala, Sweden
[3] Hong Kong University of Science and Technology, Hong Kong, China

Abstract

We study the schedulability analysis problem for non-preemptive scheduling algorithms on multiprocessors. To our best knowledge, the only known work on this problem is the test condition proposed by Baruah [1] (referred to as [BAR-EDF$_{np}$]) for non-preemptive EDF scheduling, which will reject a task set with arbitrarily low utilization if it contains a task whose execution time is equal or greater than the minimal relative deadline among all tasks. In this paper, we firstly derive a linear-time test condition which avoids the problem mentioned above, by building upon the work in [2] for preemptive multiprocessor scheduling. This test condition works on not only non-preemptive EDF, but also any other work-conserving non-preemptive scheduling algorithms. Then we improve the analysis and present test conditions of pseudo-polynomial time-complexity for Non-preemptive Earliest Deadline First scheduling (EDF$_{np}$) and Non-preemptive Fixed Priority scheduling (FP$_{np}$) respectively. Experiments with randomly generated task sets show that our proposed test conditions, especially the improved test conditions, have significant performance improvements compared with [BAR-EDF$_{np}$].

1 Introduction

Compared with preemptive scheduling, non-preemptive scheduling and schedulability analysis have received considerable less attention in the research community. However, non-preemptive scheduling is widely used in industry practice, and it may be preferable to preemptive scheduling for a number of reasons [3]: Non-preemptive scheduling algorithms are easier to implement and have lower runtime overhead than preemptive scheduling algorithms; the overhead of preemptive scheduling algorithms is more difficult to characterize and predict than that of non-preemptive scheduling algorithms due to inter-task interference caused by caching and pipelining. These benefits of non-preemptive scheduling are even more important on multiprocessor platforms, since the task migration overhead is higher and more difficult to predict. However, this problem is much less severe for non-preemptive scheduling, where each task instance runs to completion on one processor, and task migration only happens at task instance boundaries.

Non-preemptive scheduling is considered inferior for time critical systems (partially) because of its poor responsiveness. On a single processor platform, the most urgent task may not get the processor for quite a long time due to non-preemptive blocking, which could be deadly harmful for hard real time systems. However, the natural parallelism of multiprocessors can mitigate the penalty of non-preemptive blocking. Even if several processors are longtime occupied by tasks with large execution time, urgent tasks can execute on other processors. We have conducted simulation experiments to compare the performance of two well-known preemptive scheduling algorithms EDF and DM, and their non-preemptive versions EDF$_{np}$ and DM$_{np}$ on multiprocessors (details shown in Section 9). To our surprise, for task sets in which the range of task execution time is not very wide, the performance of EDF$_{np}$ is very close to EDF, and DM$_{np}$ actually performs better than DM. Note that we even have not accounted the context switch overhead in the simulations. So we believe that, for a considerable part of real-life applications on multiprocessor platforms, non-preemptive scheduling could be a better choice with respect to real time performance. This is yet another motivation for us to study non-preemptive scheduling on multiprocessors.

In this work, we study the schedulability analysis problem for sporadic task sets on identical multiprocessors with non-preemptive scheduling. We focus on *work-conserving* scheduling algorithms, i.e., it is not allowed to idle a processor if there is some task instance awaiting for execution. Note that in the context of multiprocessor scheduling, a work-conserving algorithm is necessarily a *global* [4] algorithm. The only known work on this problem is the test condition proposed by Baruah [1] (referred to as [BAR-EDF$_{np}$]) for non-preemptive EDF

scheduling (referred to as EDF$_{np}$), which has quite poor performance and suffers from the disadvantage of that task sets with arbitrarily low utilization will be rejected if $C_{max} \geq D_{min}$, where C_{max} is the maximal execution time among all tasks and D_{min} is the minimal relative deadline. In this paper we develop new sufficient schedulability tests by building upon the work in [5, 2] for preemptive scheduling algorithms. At first we derive a test condition of linear-time complexity, which works on any work-conserving non-preemptive scheduling algorithm, and can avoid the [BAR-EDF$_{np}$]'s disadvantage mentioned above. Then we derive improved test conditions for EDF$_{np}$ and FP$_{np}$ (Non-preemptive Fixed Priority scheduling) respectively, which are both of pseudo-polynomial time-complexity, but have significant performance improvement compared with [BAR-EDF$_{np}$].

The paper is structured as follows: Section 2 presents the related work. Section 3 introduces the system model and our analysis framework. We present our first general test condition in Section 4, and then improve it for EDF$_{np}$ and FP$_{np}$ in Section 5 and Section 6 respectively. The robust property of the proposed tests is proved in Section 7. Section 8 presents performance evaluation results and Section 9 use simulation to provide approximate performance comparison of preemptive and non-preemptive scheduling algorithms. Finally, conclusions are presented in Section 10.

2 Related Work

Preemptive Scheduling. All scheduling algorithms mentioned in this paragraph are within the context of "global preemptive", for example, we refer to the "global preemptive EDF" as "EDF" for short. Goossens et al. [6] introduced a schedulability test of polynomial time-complexity for periodic task sets scheduled by EDF based on the resource-augmentation techniques [7]. Similar techniques are also used in [8], to derive schedulability tests for tasks with limited utilization scheduled by RM. Baker [5] presented schedulability tests of both EDF and DM by assuming that a given task τ_k's task instance misses deadline, and then determining necessary conditions on the parameters of all the tasks to cause such a deadline miss. Based on Baker's idea, Bertogna et al. [9] observed that the work done in parallel with a task instance do not need to be accounted into its interference, and provided a new test condition of polynomial time-complexity, which can occasionally outperform Baker's test condition. Baruah [2] extended Baker's approach to reduce the over-estimation of the so-called "carry-in", and provided a test condition of pseudo-polynomial time-complexity, which has much higher acceptance ratio than previous test conditions for task systems satisfying the following conditions: the number of tasks n is significantly larger than the number of processors m (i.e., $n \gg m$), or the parameters of different tasks have widely varying orders of magnitude. Recently, Baruah et al. [10, 11, 12] have developed a new approach based on Baker's idea, which provides test conditions (of pseudo-polynomial time-complexity or polynomial time-complexity depending on different accuracy degrees) for both EDF and DM , as well as some interesting resource augmentation bounds. Andersson et al. [13] first used the approximate response time analysis for multiprocessor scheduling, which was later improved by Bertogna et al. [14] by utilizing their observation in [9] and exploring task slack time to reduce the pessimistic degree in the computation of the approximate response time. Similar to the exact response time analysis for single-processor scheduling, the time-complexity of their approaches is pseudo-polynomial.

Non-Preemptive Scheduling. For single-processor scheduling, Jeffay et al. [3] considered non-preemptive algorithms for scheduling periodic or sporadic task systems with relative deadline equal to period under the work-conserving assumption and presented a exact schedulability test of pseudo-polynomial time-complexity for periodic or sporadic task set with the EDF$_{np}$ scheduling algorithm on a single processor. George et al. [15] study general task models in which relative deadlines and periods are not necessarily related, and established exact schedulability tests for both EDF$_{np}$ and FP$_{np}$ on a single processor of pseudo-polynomial time-complexity. Recently, Baruah et al. [16] studied the schedulability analysis for non-preemptive recurring tasks, which is the general form of non-preemptive sporadic tasks, and showed that the non-preemptive schedulability analysis can be reduced to a polynomial number of preemptive schedulability analysis problems. For multiprocessor scheduling, Baruah [1] proposed a sufficient but not necessary polynomial-time schedulability test condition [BAR-EDF$_{np}$] for global EDF$_{np}$ for periodic task sets, which can be easily generalized to sporadic task sets. [BAR-EDF$_{np}$] used the technique similar to [6] and took into account the extra interference time caused by non-preemption. [BAR-EDF$_{np}$] showed that a task set τ is EDF$_{np}$ schedulable on m processors if

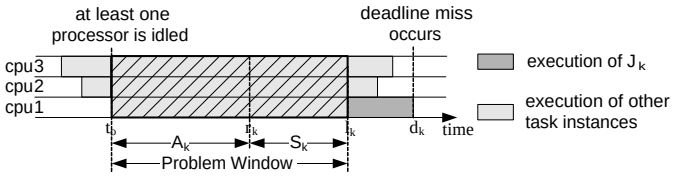
$$V_{sum}(\tau) \leq m - (m-1)V_{max}(\tau) \tag{1}$$

where

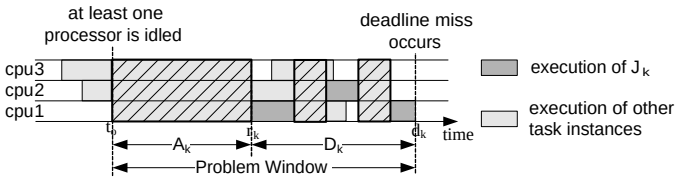
$$V_{sum}(\tau) = \sum_{\tau_i \in \tau} V_i, \quad V_{max}(\tau) = \max_{\tau_i \in \tau} V_i$$

$$V_i = \begin{cases} \frac{C_i}{D_i - C_{max}} & D_i > C_{max} \\ \infty & D_i \leq C_{max} \end{cases}$$

and C_{max} is the maximum execution time among all tasks. It is obvious that a task set with arbitrarily low utilization can not pass the test if $C_{max} \geq D_{min}$, where D_{min} denotes the minimal D_i among all tasks. Intuitively, it means that for any task instance J_k, if there is some task with execution time large enough to cover its relative deadline D_k, J_k will definitely miss its deadline, which is true for single processor scheduling, but not necessarily true for multiprocessor scheduling, since even if there are some processors longtime occupied by a task instance with large C_i, other task instances can execute on other processors to meet their deadlines.



(a) the problem window in non-preemptive scheduling



(b) the problem window in preemptive scheduling

Figure 1. the problem window in preemptive and non-preemptive scheduling.

3 System Model and Analysis Framework

We adopt the discrete time model in this paper, i.e., any time value t involved in scheduling and any task parameter is assumed to be a non-negative integer value.

We assume that a multiprocessor platform consisting of m identical processors. A sporadic task set τ consists of n sporadic tasks. A sporadic task is denoted by $\tau_i = (C_i, D_i, T_i)$, where C_i is the worst-case execution time, D_i is the relative deadline and T_i is the minimum inter-release separation, which is also referred to as the period of the task, and we assume $D_i \leq T_i$. We define $S_i = D_i - C_i$. The *utilization* of task τ_i is defined as $U_i = \frac{C_i}{T_i}$, and we use $U(\tau)$ to denote the sum of U_i of all $\tau_i \in \tau$.

Such a sporadic task τ_i generates a potentially infinite sequence of *task instances* (also known as *jobs*) with successive releases separated by at least T_i time units. We use J_i^p to denote the p^{th} instances of τ_i. We also use J_i to denote τ_i's instance in general if we do not want to specify which instance it is. Each task instance has a release time (arrival time) r_i and a absolute deadline $d_i = r_i + D_i$. We use $l_i = d_i - C_i$ to denote the latest start time of task instance J_i, i.e., if J_i starts execution after l_i, it must miss its deadline.

For fixed priority scheduling, we use $P(\tau_i)$ to denote τ_i's priority. We assume that every task has a unique priority in our task system, and use $P(\tau_i) > P(\tau_j)$ to denote that τ_i's priority is higher than τ_j's.

We aim to analysis the schedulability of a sporadic task set on multiprocessors with non-preemptive scheduling. As shown in [3], exact feasibility-analysis of periodic task systems upon non-preemptive systems is highly intractable, even upon single-processors. So our goal is to obtain sufficient, rather than exact, conditions for schedulability test.

In the following we will introduce the general framework of our analysis, which is closely related to the work in [2] for preemptive scheduling. Suppose a task set τ is non-schedulable, and let J_k be the first task instance that misses deadline. Let t_o denote the latest time-instant earlier than r_k at which at least one processor is idle and let $A_k = r_k - t_o$. Since all processors are idle when the system starts, so there always exists such a t_o. Since preemption is not allowed, once a task instance starts execution, it must run to completion without interruption. So if J_k starts to execute before its latest start time l_k, it must be able to finish execution before deadline d_k. Therefore, *in order for J_k to miss its deadline, all m processors must be continuously busy in the time interval $[t_o, l_k]$*. What happens after l_k has no effect on the schedulability of J_k. We name the time interval $[t_o, l_k]$ as **problem window**, as shown in Figure 1-(a).

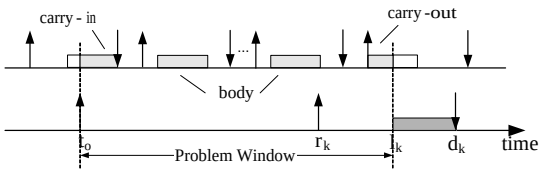


Figure 2. body, carry-in and carry-out of a task in the problem window $[t_o, l_k]$.

The definition of problem window here is different from the one in [2] for preemptive scheduling, where all m processors must be continuously busy in the time interval $[t_o, r_k]$, but does not have to be continuously busy in the time interval $[r_k, d_k]$ as long as the sum of the busy segments (shadow area in the figure) is large enough to cause τ_k to miss its deadline, as shown in Figure 1-(b).

The necessary condition for the deadline miss to occur is that the worst-case work done in the problem window by all other task instances in the task set τ except J_k, is larger than $(A_k + S_k) * m$ (the shadow area in Figure 1-(a)). Since there is no critical instant in multiprocessor scheduling, it is not possible to find the worst-case situation without exhaustively simulating the system. So we will compute the worst-case work done by each task in the problem window, denoted by $I(\tau_i)$, and use the sum of each $I(\tau_i)$ as an upper bound of the overall worst-case work done in the problem window. The work done by of a task τ_i in the problem window can be categorized into three types:

- *body*: the contribution of all task instances (called body instance) with both release time and deadline in the problem window; each task instance contributes to the workload in that interval with a complete execution time C_k;
- *carry-in*: the contribution of at most one task instance (called carry-in instance) with release time earlier than t_o and deadline in the problem window; this task instance contributes with the fraction of its execution time actually executed in the problem window.
- *carry-out*: the contribution of at most one task instance (called carry-out instance) with release time in the problem window and deadline later than l_k; this task instance

contributes with the fraction of its execution time actually executed in the problem window.

We always consider the work of a carry-in instance is executed as late as possible and a carry-out instance is executed as early as possible, as shown in Figure 2. This is a pessimistic but safe approximation to account the work done by a task.

Since there is at least one processor idled at t_o, so at most $m-1$ tasks may cause the carry-in, and the remaining $(n-m+1)$ tasks has no carry-in. We use $I_1(\tau_i)$ to denote $I(\tau_i)$ if τ_i has no carry-in instance, and use $I_2(\tau_i)$ to denote $I(\tau_i)$ if τ_i has a carry-in instance, and define:

$$I_{df}(\tau_i) = I_2(\tau_i) - I_1(\tau_i) \tag{2}$$

We sort all $I_{df}(\tau_i)$ in a non-increasing list, and use $\triangle_{I_{df}}^{m-1}$ to denote the sum of the first $(m-1)$ elements in this list, then we can get a sufficient condition for τ to be schedulable:

Lemma 1. *A task set τ is schedulable with work-conserving non-preemptive scheduling algorithms on m processors, if for any task τ_k and any $A_k \geq 0$ the following condition is satisfied:*

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + \triangle_{I_{df}}^{m-1} < (A_k + S_k) * m \tag{3}$$

4 The First Schedulability Test

To use Lemma 1 for schedulability test, we should compute the LHS of Inequality 3 as well as solve the unknown variable A_k in the inequality (The LHS of the inequality also implicitly contains A_k).

Simple upper bounds of $I_1(\tau_i)$ and $I_2(\tau_i)$ can be obtained by pessimistically accounting both the carry-in and carry-out of a task τ_i as C_i and accounting the work done by its body instances as $\lfloor \frac{A_k+S_k}{T_i} \rfloor C_i$. At the same time, we observed that as A_k increases, the proportion of the carry-in and carry-out in the overall work done by a task in the problem window tends to decrease, which implies that the adverse effect of the over-estimation of the carry-in and carry-out is more severe with small $A_k + S_k$ values, of which the extreme case is $A_k = 0$. We get our first test condition based on the observations above:

Theorem 1. [TEST-1] *A task set τ is schedulable with work-conserving non-preemptive scheduling algorithms on m processors if:*

$$U(\tau) < m - \frac{\sum_i^n C_i + \triangle_{C_i}^{m-1}}{S_{min}} \tag{4}$$

where $\sum_i^n C_i$ is the sum of all tasks' C_i, S_{min} is the minimal S_i among all tasks; we sort all C_i in a non-increasing list, and use $\triangle_{C_i}^{m-1}$ to denote the sum of the first $(m-1)$ elements in this list,

Proof. We prove the theorem by contradiction. Assume a task set τ satisfies Inequality 4 but it is non-schedulable, and with a task τ_k missing its deadline.

Since τ_k is non-schedulable, by Lemma 1 we have:

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + \triangle_{I_{df}}^{m-1} \geq (A_k + S_k) * m \tag{5}$$

The number of τ_i's body instances in the problem window is at most $\lfloor \frac{A_k+S_k}{T_i} \rfloor$, and the carry-in and the carry-out are both at most C_i, so by the definition of $I_1(\tau_i)$ and $I_2(\tau_2)$, we have:

$$I_1(\tau_i) \leq \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i + C_i \leq \frac{A_k + S_k}{T_i} * C_i + C_i \quad \Rightarrow$$

$$I_1(\tau_i) \leq (A_k + S_k)U_i + C_i \tag{6}$$

and

$$I_2(\tau_i) \leq \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i + 2C_i \leq \frac{A_k + S_k}{T_i} * C_i + 2C_i \quad \Rightarrow$$

$$I_2(\tau_i) \leq (A_k + S_k)U_i + 2C_i \tag{7}$$

so $I_{df}(\tau_i) = C_i$, so we have $\triangle_{I_{df}}^{m-1} = \triangle_{C_i}^{m-1}$.

Therefore, we have

$$(A_k + S_k)U(\tau) + \sum_i^n C_i + \triangle_{C_i}^{m-1} \geq (A_k + S_k) * m \tag{8}$$

and since $A_k \geq 0$ and $S_k \geq S_{min}$, we get

$$U(\tau) \geq m - \frac{\sum_i^n C_i + \triangle_{C_i}^{m-1}}{S_{min}}$$

which contradicts our assumption that τ satisfies Inequality 4. □

Note that [TEST-1] does not suffer from the disadvantage in [BAR-EDF$_{np}$] that any task set with $C_{max} \geq D_{min}$ will be rejected.

[TEST-1] works on any work-conserving non-preemptive scheduling algorithm, since it does not rely on any scheduling algorithm-specific property except requiring that no processor can be idle if there is some task instance awaiting for execution.

U_i, $\sum_i^n C_i$ and S_{min} all can be computed in linear time, and we can use linear-time selection [17] to compute $\triangle_{C_i}^{m-1}$, so [TEST-1] is with linear-time complexity, which is the same as [BAR-EDF$_{np}$]. [TEST-1] is still deeply pessimistic, since it used very coarse bounds on $I_1(\tau_i)$ and $I_2(\tau_i)$. In Section 5 and 6, we will present less-pessimistic test conditions for EDF$_{np}$ and FP$_{np}$ by deriving more precise bounds on $I_1(\tau_i)$ and $I_2(\tau_i)$, where we assume each task τ_i exactly executes for C_i and its instances are exactly released with separations of T_i, and later in Section 7 we will show the test conditions are still correct if this assumption is broken.

5 The Improved Test for EDF$_{np}$

In last section, we pessimistically assume that every carry-out instance contributes to the overall work in the problem window. Actually, a carry-out instance can execute in the problem

window only if it can interfere with J_k, otherwise, it must execute after l_k. Now we discuss the possible interference on a task instance J_k in EDF$_{np}$. We assume the priority ties are broken arbitrarily in the EDF$_{np}$ scheduler.

Lemma 2. *For EDF$_{np}$, if $D_i > D_k$, the necessary condition for J_i to cause interference to J_k is $r_i < r_k$, i.e., J_i must be released earlier than J_k; if $D_i \leq D_k$, the necessary condition for J_i to cause interference to J_k is $d_i \leq d_k$, i.e., J_i's absolute deadline must be no later than that of J_k.*

Proof. There are two types of interference in non-preemptive scheduling:

1. Interference caused by the priority order. A task instance J_i can cause this kind of interference to J_k only if $d_i \leq d_k$. Note that since we assume the priority ties are arbitrarily broken, J_i may interfere with J_k if $d_i = d_k$.

2. Interference caused by non-preemption blocking. A task instance J_i can cause this kind of interference to J_k only if J_i has already been running when J_k is released, which implies $r_i < r_k$.

If $D_i > D_k$, suppose $r_i \geq r_k$, then J_i can not cause the second type of interference; since $D_i > D_k$, J_i's deadline must be later than d_k, so it also can not cause the first type of interference. If $D_i \leq D_k$, suppose $d_i > d_k$, then J_i can not cause the first type of interference; since $D_i \leq D_k$, J_i's release time must be later than r_k, so it also can not cause the second type of interference. □

5.1 Computing $I_1(\tau_i)$ for EDF$_{np}$

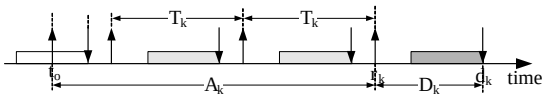


Figure 3. the worst case of $I_1(\tau_i)$ if $i = k$.

At first we compute $I_1(\tau_i)$ with $i = k$, i.e., the worst-case work done by τ_k's body instances. As shown in Figure 3, the number of τ_k's body instances is $\lfloor \frac{A_k}{T_k} \rfloor$, so we have

$$I_1^1(\tau_i) = \lfloor \frac{A_k}{T_k} \rfloor * C_k \tag{9}$$

Next we will compute $I_1(\tau_i)$ with $i \neq k$. The following Lemma shows the worst case of $I_1(\tau_i)$ of a task τ_i with $i \neq k$.

Lemma 3. *The worst case of $I_1(\tau_i)$ ($i \neq k$) occurs when one of τ_i's instances is released at the time-instant t_o.*

Proof. To prove the worst case of $I_1(\tau_i)$ occurs when one of τ_i's instances is released at the time-instant t_o, we should prove that based on this case, moving all τ_i's releases rightwards for a distance x ($x < T_i$) will not increase $I_1(\tau_i)$[1].

[1]Moving τ_i's releases for any distance can be transformed to an equal form of moving them rightwards (or leftwards) for x ($x < T_i$).

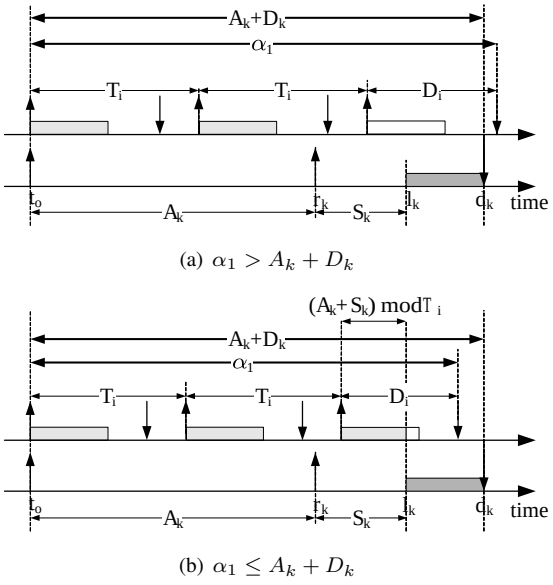


Figure 4. the worst case of $I_1(\tau_i)$ if $D_i \leq D_k$.

We use J_i^p to denote the first body instance of τ_i. At the left end of the problem window, J_i^{p-1}'s release time is $t_o - T_i + x$ after being moved rightwards for x. Since $x < T_i$, J_i^{p-1}'s release time is still earlier than t_o, and by the definition of $I_1(\tau_i)$, J_i^{p-1} has no contribution to $I_1(\tau_i)$ after being moved, so there is no increase to $I_1(\tau_i)$ at the left end of the problem window. At the right end of the problem window, trivially there is no increase to $I_1(\tau_i)$. So moving all τ_i's releases rightwards for a x ($x < T_i$) will not increase $I_1(\tau_i)$. □

Now we compute $I_1(\tau_i)$ ($i \neq k$) in this worst case:

1. $D_i \leq D_k$. By Lemma 2, we know a task instance of τ_i with $D_i \leq D_k$ can interfere with J_k only if its deadline is no later than d_k. We use α_1 to denote the distance between t_o and the deadline of the τ_i's last instance released before l_k:

$$\alpha_1 = \lfloor \frac{A_k + S_k}{T_i} \rfloor * T_i + D_i \tag{10}$$

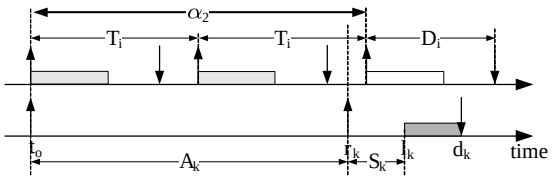
   (a) $\alpha_1 > A_k + D_k$. As shown in Figure 4(a), the deadline of τ_i's last instance released in the problem window is later than d_k, so it has no contribution to $I_1(\tau_i)$. The number of τ_i's instances contributing to $I_1(\tau_i)$ is $\lfloor \frac{A_k+S_k}{T_i} \rfloor$. So we have:

$$I_1^2(\tau_i) = \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i \tag{11}$$

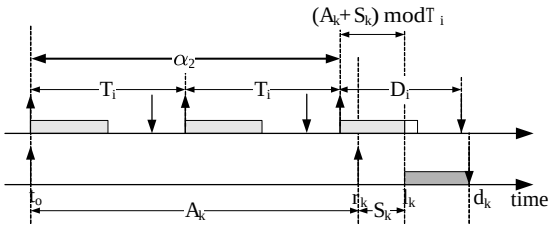
   (b) $\alpha_1 \leq A_k + D_k$. As shown in Figure 4(b), the deadline of τ_i's last instance released in the problem window is no later than d_k, so it contributes to $I_1(\tau_i)$, and the contribution is bounded by both C_i

and $(A_k + S_k) \bmod T_i$. So we have:

$$I_1^3(\tau_i) = \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i + \min(C_i, (A_k + S_k) \bmod T_i) \quad (12)$$



(a) $\alpha_2 \geq A_k > 0$



(b) $\alpha_2 < A_k$

Figure 5. the worst case of $I_1(\tau_i)$ if $D_i > D_k$.

2. $D_i > D_k$ By Lemma 2, we know an instance of τ_i with $D_i > D_k$ can interfere with J_k only if its release time is earlier than r_k. We use α_2 to denote the distance between t_o and the release time of τ_i's last instance released in the problem window:

$$\alpha_2 = \lfloor \frac{A_k + S_k}{T_i} \rfloor * T_i \quad (13)$$

   (a) $A_k = 0$. If $A_k = 0$, then $t_o = r_k$. Since $D_i > D_k$, any task instance released no earlier than t_o has deadline later than d_k, so can not interfere with J_k. So in this case $I_1(\tau_i) = 0$.

   (b) $\alpha_2 \geq A_k > 0$. As shown in Figure 5(a), the release time of τ_i's last instance released in the problem window is no earlier than r_k, so it can not interfere with J_k. The number of τ_i's instances contributing to $I_1(\tau_i)$ is $\lfloor \frac{A_k + S_k}{T_i} \rfloor$. So in this case $I_1(\tau_i)$ is computed by Equation 11.

   (c) $\alpha_2 < A_k$. As shown in Figure 5(b), the release time of τ_i's last instance released in the problem window is earlier than r_k, so it contributes to $I_1(\tau_i)$, and its contribution is bounded by both C_i and $(A_k + S_k) \bmod T_i$. So in this case $I_1(\tau_i)$ is computed by Equation 12.

By the discussions above, we can compute $I_1(\tau_i)$ for EDF$_{np}$ by:

$$I_1(\tau_i) = \begin{cases} 0 & D_i > D_k \wedge A_k = 0 \\ I_1^1(\tau_i) & i = k \\ I_1^2(\tau_i) & (i \neq k \wedge D_i \leq D_k \wedge \alpha_1 > A_k + D_k) \\ & \vee (D_i > D_k \wedge \alpha_2 \geq A_k > 0) \\ I_1^3(\tau_i) & otherwise \end{cases} \quad (14)$$

where $I_1^1(\tau_i)$, $I_1^2(\tau_i)$, $I_1^3(\tau_i)$, α_1 and α_2 are defined in Equation 9, 11, 12, 10 and 13 respectively.

5.2 Computing $I_2(\tau_i)$ for EDF$_{np}$

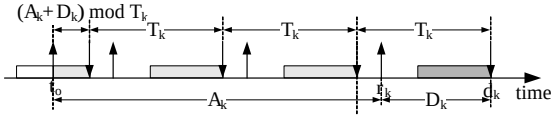


Figure 6. the worst case of $I_2(\tau_i)$ if $i = k$.

At first we compute $I_2(\tau_i)$ with $i = k$, i.e., the worst-case woke done by τ_k's carry-in and body instances. As shown in Figure 6, if we take the interval between the deadlines of two adjoining instances of τ_i as a "unit", there are $\lfloor \frac{A_k + D_k}{T_k} \rfloor$ such "units" in the time interval $[t_o, d_k]$. The carry-in is bounded by both C_k and $(A_k + D_k) \bmod T_k$. At the same time, the work done by J_k itself should be subtracted. So for τ_i with $i = k$, we have:

$$I_2^1(\tau_i) = \lfloor \frac{A_k + D_k}{T_k} \rfloor * C_k + \min(C_k, (A_k + D_k) \bmod T_k) - C_k \quad (15)$$

In the following we will compute $I_2(\tau_i)$ with $i \neq k$ in Lemma 4, 5 and 6.

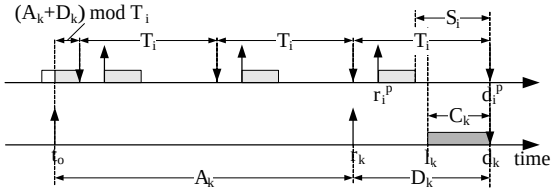


Figure 7. the worst case of $I_2(\tau_i)$ if $D_i \leq D_k \wedge S_i > C_k$.

Lemma 4. *The worst case of $I_2(\tau_i)$ $(i \neq k)$ occurs when one of τ_i's instance has its deadline at d_k, if*

$$D_i \leq D_k \wedge S_i > C_k \quad (16)$$

and in this case we can compute $I_2(\tau_i)$ by:

$$I_2^2(\tau_i) = \lfloor \frac{A_k + D_k}{T_i} \rfloor * C_i + \min(C_i, (A_k + D_k) \bmod T_i) \quad (17)$$

Proof. Let J_i^p be the instance with its deadline at d_k, and by Lemma 2, we know J_i^p may interfere with J_k. As shown in Figure 7, since $S_i > C_k$, J_i^p's contribution to $I_2(\tau_i)$ is C_i.

Now we examine whether $I_2(\tau_i)$ will be increased if we move all τ_i's releases leftwards for x ($x < T_i$).

After moving leftwards for x ($x < T_i$), at the right end of the problem window, the contribution of J_i^p is still C_i; the deadline of J_i^{p+1} is at $d_k + T_i - x$, and since $x < T_i$, it is still later than d_k, so has no contribution to $I_2(\tau_i)$, so $I_2(\tau_i)$ will not be increased at the right end of the problem window. At the left end, trivially there is no increased to $I_2(\tau_i)$. So $I_2(\tau_i)$ will not be increased after moving τ_i's releases leftwards for x ($x < T_i$), so $d_i^p = d_k$ is the worst case for $I_2(\tau_i)$ for tasks with $D_i \leq D_k \wedge S_i > C_k$.

As shown in Figure 7, we take the interval between the deadlines of two adjoining instances of τ_i as a "unit", then there are $\lfloor \frac{A_k + D_k}{T_k} \rfloor$ such "units" in the time interval $[t_o, d_k]$. The carry-in is bounded by both C_i and $(A_k + D_k) \bmod T_i$. So we can compute $I_2(\tau_i)$ by Equation 17. □

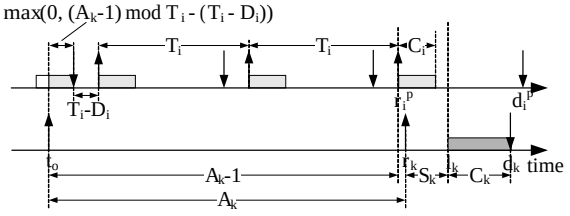


Figure 8. the worst case of $I_1(\tau_i)$ if $D_i > D_k \wedge S_k \geq C_i$.

Lemma 5. *The worst case of $I_2(\tau_i)$ ($i \neq k$) occurs when one of τ_i's instance is released at $r_k - 1$, if*

$$D_i > D_k \wedge S_k \geq C_i \tag{18}$$

and in this case we can compute $I_2(\tau_i)$ by:

$$I_2^3(\tau_i) = \begin{cases} C_i - 1 & A_k = 0 \\ (\lfloor \frac{A_k - 1}{T_i} \rfloor + 1) * C_i + \upsilon & A_k > 0 \end{cases} \tag{19}$$

where

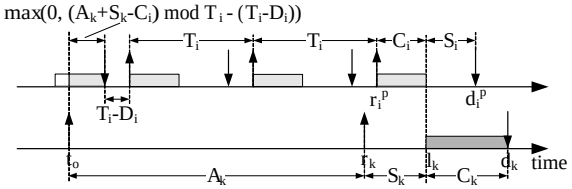
$$\upsilon = \min(C_i, \max(0, (A_k - 1) \bmod T_i - (T_i - D_i))) \tag{20}$$

Proof. Let J_i^p be τ_i's task instance released at $r_k - 1$, as shown in Figure 8. By Lemma 2 we know J_i^p can interfere with J_k. Since $S_k \geq C_i$, the contribution of J_i^p is C_i.

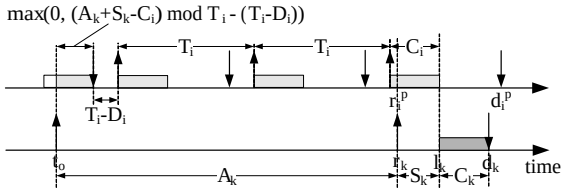
Now we examine whether $I_2(\tau_i)$ will be increased if we move all τ_i's releases leftwards for x ($x < T_i$).

After moving leftwards for x ($x < T_i$), at the right end of the problem window, the contribution of J_i^p is still C_i; the release time of J_i^{p+1} is $r_k - 1 + T_i - x$, since $x < T_i$, it is still not earlier than r_k, so J_i^{p+1} has no contribution to $I_2(\tau_i)$. Therefore, $I_2(\tau_i)$ will not be increased at the right end of the problem window. At the left end, trivially the contribution will not be increased. So $I_2(\tau_i)$ will not be increased after moving τ_i's releases leftwards for x ($x < T_i$), so $r_i^p = r_k - 1$ is the worst case for $I_2(\tau_i)$ for tasks with $D_i > D_k \wedge S_k \geq C_i$.

If $A_k > 0$, as shown in Figure 8, the number of τ_i's body instances is $\lfloor \frac{A_k - 1}{T_i} \rfloor$, the carry-out is C_i, the carry-in is bounded by both C_i and the distance between t_o and the deadline of the carry-in instance, which equals to $\max(0, (A_k - 1) \bmod T_i - (T_i - D_i))$. If $A_k = 0$, $I_2(\tau_i) = C_i - 1$. So we can compute $I_2(\tau_i)$ by Equation 19. □



(a) $D_i \leq D_k \wedge S_i \leq C_k$



(b) $D_i > D_k \wedge S_k < C_i$

Figure 9. the worst case of $I_2(\tau_i)$ if $D_i \leq D_k \wedge S_i \leq C_k$ or $D_i > D_k \wedge S_k < C_i$.

Lemma 6. *The worst case of $I_2(\tau_i)$ ($i \neq k$) occurs when one of τ_i's instances is released at $l_k - C_i$, if*

$$(D_i \leq D_k \wedge S_i \leq C_k) \vee (D_i > D_k \wedge S_k < C_i) \tag{21}$$

and in this case we can compute $I_2(\tau_i)$ by:

$$I_2^4(\tau_i) = \begin{cases} A_k + S_k & A_k + S_k \leq C_i \\ \lfloor \frac{A_k + S_k - C_i}{T_i} \rfloor * C_i + C_i + \omega & A_k + S_k > C_i \end{cases} \tag{22}$$

where

$$\omega = \min(C_i, \max(0, (A_k + S_k - C_i) \bmod T_i - (T_i - D_i))) \tag{23}$$

Proof. At first we will show that if $r_i^p = l_k - C_i$, J_i^p contributes C_i to $I_2(\tau_i)$:

- $D_i \leq D_k \wedge S_i \leq C_k$. As shown in Figure 9(a), since $S_i \leq C_k$, we have $d_i^p \leq d_k$, so J_i^p can interfere with J_k, and since $S_i \leq C_k$, the contribution of J_i^p is C_i.

- $D_i > D_k \wedge S_k < C_i$. As shown in Figure 9(b), since $S_k \leq C_i$, we have $r_i^p \leq r_k$, so J_i^p can interfere with J_k, and since $S_k \leq C_i$, the contribution of J_i^p is C_i.

Now we examine whether $I_2(\tau_i)$ will be increased if we move all τ_i's releases leftwards for x ($x < T_i$):

- If we move τ_i's releases leftwards for x_1 ($x_1 < T_i - C_i$), at the left end of the problem window, trivially the contribution can not be increased; at the right end of the problem window, the contribution of J_i^p is still C_i, the release time of the J_i^{p+1} is $l_k+(T_i-C_i)-x_1$, and since $x_1 < T_i-C_i$, it is still later than l_k after moving, so J_i^{p+1} has contribution to $I_2(\tau_i)$. Therefore there is no increase of $I_2(\tau_i)$ at the right end of the problem window. So moving τ_i's release time leftwards for x_1 ($x_1 < T_i - C_i$) will not increase $I_2(\tau_i)$.

- Moving τ_i's releases leftwards for x_2 ($T_i - C_i \leq x_2 < T_i$) has the same effect as moving τ_i's release time rightwards for $T_i - x_2$. Since x_2 ($T_i - C_i \leq x_2 < T_i$), we have $0 < T_i - x_2 \leq C_i$. So at the right end of the problem window, the contribution of J_i^p is decreased by $T_i - x_2$ after being moved rightwards for $T_i - x_2$; at the left end of the problem window, $I_2(\tau_i)$ is at most increased by $T_i - x_2$. So $I_2(\tau_i)$ will not be increased after moving τ_i's releases rightwards for $T_i - x_2$ ($T_i - C_i \leq x_2 < T_i$).

In summary, $I_2(\tau_i)$ will not be increased if we move τ_i's releases leftwards for x ($x < T_i$). So $r_i^p = l_k - C_i$ is the worst case for $I_2(\tau_i)$.

If $A_s + S_k \leq C_i$, it is easy to see $I_2(\tau_i) = A_k + S_k$. If $A_s + S_k > C_i$, the number of body instance is $\lfloor \frac{A_k+S_k-C_i}{T_i} \rfloor$; the carry-out is C_i; the carry-in is bounded by both C_i and the distance between t_o and the deadline of the carry-in instance, which equals to $\max(0, (A_k + S_k - C_i) \bmod T_i - (T_i - D_i))$. So we can compute $I_2(\tau_i)$ by Equation 22. □

By the discussions above, we can compute $I_2(\tau_i)$ for EDF$_{np}$ by:

$$I_2(\tau_i) = \begin{cases} I_2^1(\tau_i) & i = k \\ I_2^2(\tau_i) & i \neq k \wedge D_i \leq D_k \wedge S_i > C_k \\ I_2^3(\tau_i) & D_i > D_k \wedge S_k \geq C_i \\ I_2^4(\tau_i) & \text{otherwise} \end{cases} \quad (24)$$

where $I_2^1(\tau_i)$, $I_2^2(\tau_i)$, $I_2^3(\tau_i)$ and $I_2^4(\tau_i)$ are defined in Equation 15, 17, 19, 22 respectively.

5.3 A New Test Condition for EDF$_{np}$

By now we have obtained a sufficient schedulability test condition for EDF$_{np}$ by Lemma 1 and the computation of $I_1(\tau_i)$ and $I_2(\tau_i)$ above:

Theorem 2. [TEST-EDF$_{np}$] *A task set is EDF$_{np}$ schedulable on m identical processors if for any task τ_k and for any A_k we have:*

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + \triangle_{I_{df}}^{m-1} < (A_k + S_k) * m \quad (25)$$

where $I_1(\tau_i)$ and $I_2(\tau_i)$ are defined in Equations14 and 24.

For any given τ_k and A_k, the LHS of Condition 25 can be evaluated in time linear with n, since the time for computing $I_1(\tau_i)$, $I_2(\tau_i)$ and $I_{\text{df}}(\tau_i)$ for each τ_i is $O(n)$, and we can use linear-time selection [17] to compute $\triangle_{I_{df}}^{m-1}$. The next theorem tells us the range of A_k that should be tested:

Theorem 3. *For any task set with $U(\tau) < m$, if condition 25 is to be violated for any A_k, then it is violated for some A_k that satisfies the condition below:*

$$A_k \leq \frac{\sum_i^n C_i + \triangle_{C_i}^{m-1}}{m - U(\tau)} - S_k \quad (26)$$

Proof. As shown in Section 4 (Inequality 6, 7), we know:

$$I_1(\tau_i) \leq (A_k + S_k) * U_i + C_i$$

$$I_2(\tau_i) \leq (A_k + S_k) * U_i + 2C_i$$

If Condition 25 is violated, it must be true that

$$\sum_i^n C_i + \triangle_{C_i}^{m-1} + (A_k + S_k)U(\tau) \geq (A_k + S_k)m$$

$$\Rightarrow \quad A_k \leq \frac{\sum_i^n C_i + \triangle_{C_i}^{m-1}}{m - U(\tau)} - S_k$$

□

Since A_k is a non-negative integer, Condition 25 can be checked in time pseudo-polynomial to the task parameters, for all task systems τ for which $U(\tau)$ is bounded by a constant strictly less than the number of processors m.

As mentioned in Section 4, the effect of the over-estimation of the carry-in and carry-out is more severe with smaller $A_k + S_k$ values. So one should check the tasks in increasing order of their S_ks, and in increasing order of A_ks for each task, to advance the testing efficiency. We have tested task sets consisting of one hundred of tasks with each T_i uniformly distributed in $[10, 2000]$. The tests of 1,000,000 such task sets are finished in several minutes. We believe this rates suggest that [TEST-EDF$_{np}$] is not only applicable to off-line schedulability test, but also a good candidate for on-line admission control with moderate-scale task systems.

6 The Improved Test for FP$_{np}$

The following lemma shows the possible interference on caused by lower-priority tasks in FP$_{np}$.

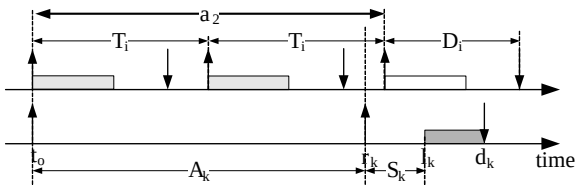
Lemma 7. *A task instance J_i with $P(\tau_i) < P(\tau_k)$ can interfere with J_k of τ_k only if J_i is released before r_k.*

6.1 Computing $I_1(\tau_i)$ for FP$_{np}$

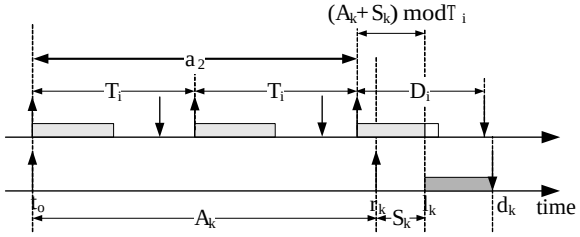
If $i = k$, the computation of $I_1(\tau_i)$ is the same as in Equation 9:

$$I_1^1(\tau_i) = \lfloor \frac{A_k}{T_k} \rfloor * C_k$$

Now we compute $I_1(\tau_i)$ with $i \neq k$. The conclusion of Lemma 3 also works for FP$_{np}$, i.e., the worst case of $I_1(\tau_i)$ occurs when one of τ_i's instance is released at t_o. Now we compute $I_1(\tau_i)$ in this worst case.



(a) $\alpha_2 \geq A_k$



(b) $\alpha_2 < A_k$

Figure 10. $P(\tau_i) < P(\tau_k)$**.**

1. $P(\tau_i) < P(\tau_k)$ We still use α_2 to denote the distance between t_o and the release time of τ_i's last instance released in the problem window:

$$\alpha_2 = \lfloor \frac{A_k + S_k}{T_i} \rfloor * T_i$$

   (a) $A_k = 0$. If $A_k = 0$, then $t_o = r_k$. Since $P(\tau_i) < P(\tau_k)$, any instance released at or later than t_o can not interfere with J_k. So we know in this case $I_1(\tau_i) = 0$.

   (b) $\alpha_2 \geq A_k > 0$. In this case, the release time of the last instance released in the problem window is not earlier than r_k, so by Lemma 7 we know it can not interfere with J_k, so we can compute $I_1(\tau_i)$ by Equation 11.

$$I_1^2(\tau_i) = \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i$$

   (c) $\alpha_2 < A_k$. In this case, the release time of the last instance that released in the problem window is earlier than r_k, so by Lemma 7 we know it can interfere with J_k, so we can compute $I_1(\tau_i)$ by Equation 12.

$$I_1^3(\tau_i) = \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i + \min(C_i, (A_k + S_k) \bmod T_i)$$

2. $P(\tau_i) > P(\tau_k)$. If $P(\tau_i) > P(\tau_k)$, any τ_i's instance released in the problem window can contribute to $I_1(\tau_i)$. So we can compute $I_1(\tau_i)$ by Equation 12.

$$I_1^3(\tau_i) = \lfloor \frac{A_k + S_k}{T_i} \rfloor * C_i + \min(C_i, (A_k + S_k) \bmod T_i)$$

So we have :

$$I_1(\tau_i) = \begin{cases} 0 & P(\tau_i) < P(\tau_k) \wedge A_k = 0 \\ I_1^1(\tau_i) & i = k \\ I_1^2(\tau_i) & P(\tau_i) < P(\tau_k) \wedge \alpha_2 \geq A_k > 0 \\ I_1^3(\tau_i) & \text{otherwise} \end{cases} \quad (27)$$

6.2 Computing $I_2(\tau_i)$ for FP$_{np}$

When $i = k$, the computation of $I_2(\tau_i)$ is the same as Equation 15.

$$I_2^1(\tau_i) = \lfloor \frac{A_k + D_k}{T_k} \rfloor * C_k + \min(C_k, (A_k + D_k) \bmod T_k) - C_k$$

In the following we will compute $I_2(\tau_i)$ with $i \neq k$ in Lemma 8 and 9. Their proofs are similar with the proofs in Section 5.2.

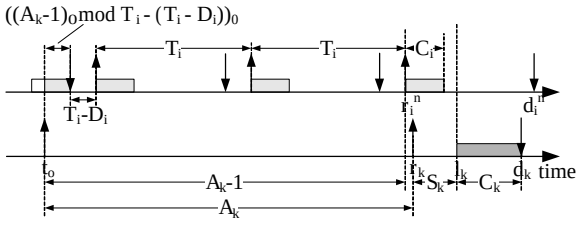


Figure 11. $P(\tau_i) < P(\tau_k) \wedge S_k \geq C_i$**.**

Lemma 8. *For task with $P(\tau_i) < P(\tau_k) \wedge S_k \geq C_i$, the worst case of $I_2(\tau_i)$ occurs when one of τ_i's instances is released at the time-instant $r_k - 1$. We can compute $I_2(\tau_i)$ by Equation 19:*

$$I_2^3(\tau_i) = \begin{cases} C_i - 1 & A_k = 0 \\ (\lfloor \frac{A_k - 1}{T_i} \rfloor + 1) * C_i + \upsilon & A_k > 0 \end{cases}$$

where

$$\upsilon = \min(C_i, \max(0, (A_k - 1) \bmod T_i - (T_i - D_i)))$$

Lemma 9. *For tasks with $P(\tau_i) > P(\tau_k)$ or $P(\tau_i) < P(\tau_k) \wedge C_i > S_k$, the worst case of $I_2(\tau_i)$ occurs when one of the τ_i's instance is released at the time-instant $l_k - C_i$. We can compute $I_2(\tau_i)$ by Equation 22:*

$$I_2^4(\tau_i) = \begin{cases} A_k + S_k & A_k + S_k \leq C_i \\ \lfloor \frac{A_k + S_k - C_i}{T_i} \rfloor * C_i + C_i + \omega & A_k + S_k > C_i \end{cases}$$

where

$$\omega = \min(C_i, \max(0, (A_k + S_k - C_i) \bmod T_i - (T_i - D_i)))$$

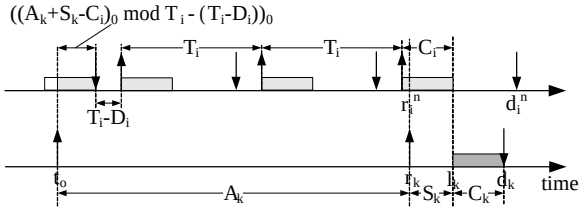


Figure 12. $P(\tau_i) > P(\tau_k)$ **or** $P(\tau_i) < P(\tau_k) \wedge C_i > S_k$

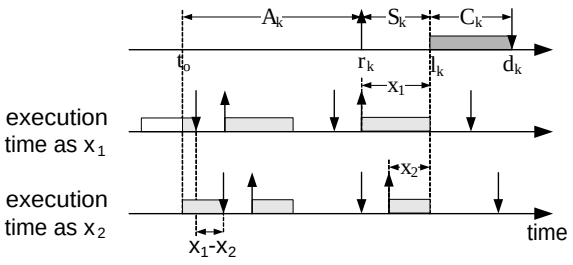


Figure 13. decreasing execution time

So we have :

$$I_2(\tau_i) = \begin{cases} I_2^1(\tau_i) & i = k \\ I_2^3(\tau_i) & P(\tau_i) < P(\tau_k) \wedge S_k \geq C_i \\ I_2^4(\tau_i) & \text{otherwise} \end{cases} \quad (28)$$

6.3 A New Test Condition for FP$_{np}$

By now we have obtained a sufficient schedulability test condition for FP$_{np}$ by Lemma 1 and the computation of $I_1(\tau_i)$ and $I_2(\tau_i)$ for FP$_{np}$ above:

Theorem 4. [TEST-FP$_{np}$] *A task set is FP$_{np}$ schedulable on m processors if for any task τ_k and for any A_k we have:*

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + \triangle_{I_{df}}^{m-1} < (A_k + S_k) * m \quad (29)$$

where $I_1(\tau_i)$ and $I_2(\tau_i)$ are computed by Equation 27 and 28 respectively.

Its computational complexity is the same as [TEST-EDF$_{np}$].

7 Robustness

A scheduling algorithm is said to be *execution time robust* (*inter-release separation robust*) if decreasing the execution times (increasing the inter-release separation) of task instances in a schedulable task set does not lead to deadline violations.

The robustness property is important. If a scheduling algorithm is robust, then the system designer only needs to consider the boundary values (C_i and T_i) to determine if the system is schedulable, rather than consider the every possible execution time in the interval [BCET, WCET] or consider the infinitely many possible inter-release time separations in $[T_i, \infty)$.

Non-preemptive scheduling of periodic tasks is neither execution time robust nor inter-release separation robust if the scheduling algorithm is work-conserving. Therefore, any exact schedulability test for work-conserving scheduling algorithms, is necessarily non-robust. However, a sufficient but not necessary schedulability test can be robust in the sense that if a sporadic task system is guaranteed by this test to meet all deadlines with their WCETs and minimal inter-release separations, then it is guaranteed to continue to meet all deadlines even if some of the execution requirements are decreased or some of the release separations are increased. In the following we will show that the tests proposed in this paper are robust.

Theorem 5. *The schedualbiility tests [TEST-EDF$_{np}$] and [TEST-FP$_{np}$] are both execution time and inter-release separation robust.*

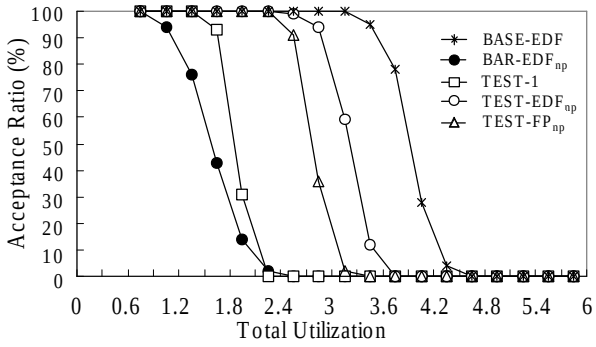
Proof. At first, when we test the schedulability of a task τ_k, we only need to consider $S_k = D_k - C_k$ in the test conditions, rather than consider every possible value of S_k. This observation is quite straightforward: if a task instance can start execution no later than l_k, it must be schedulable if it's actual execution time is smaller than C_k.

In the following, we will show when we test the schedulability of a task τ_k, it is adequate to only consider the WCET C_i and minimal inter-release separation T_i for the interfering task τ_i. It is easy to see that $I_1^1(\tau_i)$, $I_1^2(\tau_i)$, $I_1^3(\tau_i)$, $I_2^1(\tau_i)$, $I_2^2(\tau_i)$, $I_2^3(\tau_i)$ are all monotonically non-decreasing with respect to C_i and monotonically non-increasing with respect to T_i, and $I_2^4(\tau_i)$ is monotonically non-increasing with respect to T_i. The only case we will elaborate on is that $I_2^4(\tau_i)$ is monotonically non-decreasing with respect to C_i. As shown in Figure 13, when the execution time of τ_i is decreased from x_1 to x_2, τ_i's releases should be moved rightwards for $x_1 - x_2$ to keep the release time of τ_i's carry-out instance being $l_k - x_2$, so at the left end of the problem window, $I_2^4(\tau_i)$ is at most increased by $x_1 - x_2$. At the same time, the contribution of each body instance and the carry-out instance is decreased by $x_1 - x_2$, so $I_2^4(\tau_i)$ will not be increased. So $I_2^4(\tau_i)$ is monotonically non-decreasing with respect to C_i. □

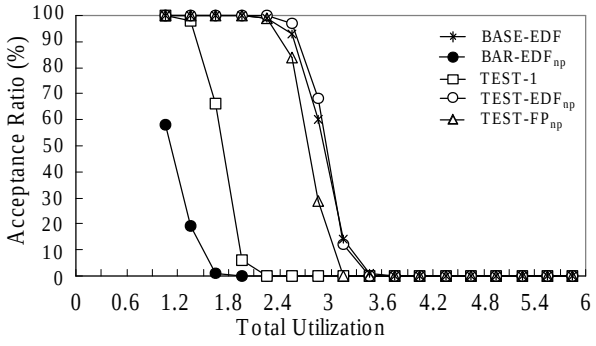
8 Performance Evaluation

The only known schedulability test condition for EDF$_{np}$ is [BAR-EDF$_{np}$], and there is no known test condition for FP$_{np}$ to our best knowledge. So we will compare [BAR-EDF$_{np}$] with our proposed test conditions [TEST-1], [TEST-EDF$_{np}$] and [TEST-FP$_{np}$]. [TEST-EDF$_{np}$] and [TEST-FP$_{np}$] are superior to [TEST-1], which means a task set accepted by [TEST-1] can also be accepted by [TEST-EDF$_{np}$] and [TEST-FP$_{np}$]. In general, our new test conditions is incomparable to [BAR-EDF$_{np}$], i.e., we can construct a task set accepted by our test conditions but rejected by [BAR-EDF$_{np}$], as well as a task set accepted by [BAR-EDF$_{np}$] but rejected by ours. In the following we will use randomly generated task sets to compare the average performances, in terms of acceptance ratio, of these

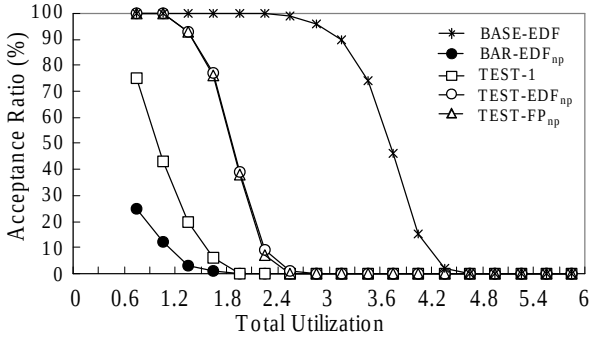
test conditions. Additionally, we will also compare the performance of the test conditions for EDF$_{np}$ with the test condition (referred as to [BASE-EDF]) in [2] for preemptive EDF, which is the base of the analysis in this paper.



(a) $0.1 \leq U_i \leq 0.4,\ 10 \leq T_i \leq 20$



(b) $0.1 \leq U_i \leq 0.6,\ 10 \leq T_i \leq 20$



(c) $0.1 \leq U_i \leq 0.4,\ 10 \leq T_i \leq 100$

Figure 14. performance comparison of the test conditions

We follow the method in [18] to generate task sets: A task set of $m+1$ tasks was generated, and tested. Then we increase the task number by 1 to generate a new task set, and all the schedulability tests were run on the new task set. This process was repeated until the total processor utilization exceeded m. The whole procedure was then repeated, starting with a new task set of $m+1$ tasks, until 1,000,000 task sets have been generated and tested. This method of generating random task sets produces a fairly uniform distribution of total utilizations, except at the extreme end of low utilization.

The task parameter setting in Figure 14(a) is as follows: The processor number is 6; for each task, T_i is uniformly distributed in $[10, 20]$, the ratio between D_i and T_i is uniformly distributed in $[0.8, 1]$, and U_i is uniformly distributed in $[0.1, 0.4]$. In this experiment, [TEST-1] performs slightly better than [BAR-EDF$_{np}$], and they are clearly outperformed by the improved test conditions. [BASE-EDF] outperforms all test conditions for non-preemptive scheduling, since more interference caused by non-preemption blocking should be taken into account in the tests for non-preemptive scheduling.

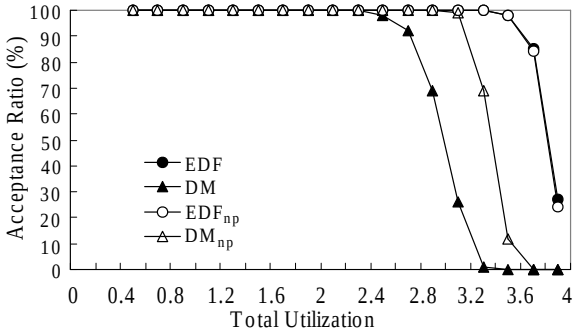
In Figure 14(b), we change the range of U_i to $[0.1, 0.6]$ and keep other settings same as in Figure 14(a). It is shown that as the average task utilization increases, the performance of [BAR-EDF$_{np}$] degrades, while the effect on our proposed test conditions is much smaller. The reason is that [BAR-EDF$_{np}$] is more sensitive to the C_{max} value. A interesting phenomena shown in this experiment is that [TEST-EDF$_{np}$] performs very close to (even a little better than) [BASE-EDF]. The reason is that the length of the problem window in the analysis of non-preemptive scheduling is C_i shorter than in the analysis of preemptive scheduling, which compensates the extra interference caused by non-preemption blocking in some degree. As discussed in Section 6.3, most of the failures in the testing occurs with small $A_k + S_k$ values, so this compensation is significant when S_k is relatively small.

In Figure 14(c), we change the range of T_i to $[10, 100]$ based on the setting in Figure 14(a), which means different tasks have a wider varying scale range. In this case, the performance of all test conditions for non-preemptive scheduling degrades rapidly, while the effect on [BASE-EDF] is trivial. This result accords with the intuition that tasks with long execution time are harmful to the schedulability of short-urgent tasks due to the non-preemptive blocking.

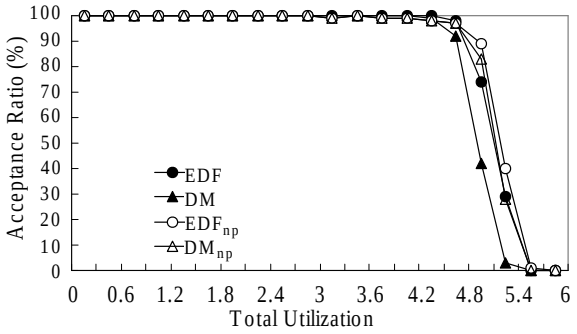
By the above experiments, we can see that the test conditions proposed in this paper, especially the improved test conditions, have a significant performance improvement compared with [BAR-EDF$_{np}$]. In general, the performance of our proposed test conditions is inferior to the test condition [BASE-EDF], which is for preemptive scheduling, while in some special cases, the performance of [TEST-EDF$_{np}$] is close to (or better than) [BASE-EDF].

9 Simulation Experiments

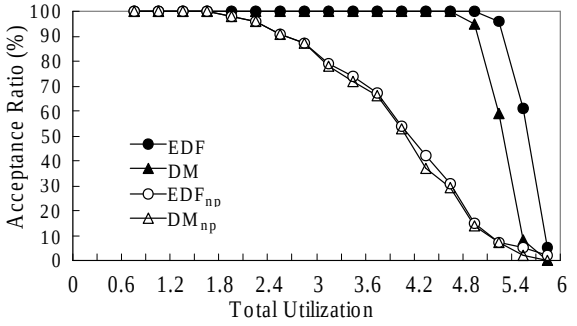
In the following, we will compare the performance of different scheduling algorithms by simulations. For hard real-time systems, the performances of the test conditions are usually considered to make more sense than the absolute performance of the scheduling algorithms. However, studying the performance characteristics of the scheduling algorithms will disclose their "potentials" and may inspirit the development of



(a) $10 \leq T_i \leq 20$



(b) $16 \leq T_i \leq 128$



(c) $16 \leq T_i \leq 512$

Figure 15. performance comparison of preemptive and non-preemptive scheduling by simulations

new scheduling and analysis techniques.

Figure 15 shows the comparison of the performance of two well-known preemptive scheduling algorithms, EDF and DM, and their non-preemptive versions, EDF$_{np}$ and DM$_{np}$. In the simulation of each task set, we set all tasks' release offsets as 0, task inter-release separation as T_i and task execution time as the C_i, and check the task set till its hyper-period. Even though these assumptions do not guarantee to generate the worst-case scenario in terms of schedulability, they are adopted since it is not computationally feasible to try all possible task release offsets, task inter-release times or task execution times. Therefore, the simulation result can at best be viewed as an approximation of the real performance of a scheduling algorithm without any guarantee of correctness.

The parameter setting of the experiment in Figure 15(a) is as follows: T_i is uniformly distributed in $[10, 30]$; the ratio between D_i and T_i is uniformly distributed in $[0.8, 1]$; U_i is uniformly distributed in $[0.1, 0.4]$. From Figure 15 (a), we can see that the performance of EDF$_{np}$ is very close to EDF, while DM$_{np}$ is actually better than DM. We also tried other priority assignment policies for fixed-priority scheduling (such as random assignment, slack time monotonic, computation time monotonic, inverse computation time monotonic etc.), and with all of them FP$_{np}$ outperforms FP more or less. Note that we even have not taken the context switch overhead into account in these simulations.

In Figure 15(b) and 15(c), we examine their performance when different tasks have wider varying scale ranges. It is not feasible to have T_i uniformly distributed in a very wide range, which may lead to very large hyper-periods. Instead, we set each task period as 2^e, where e is uniformly distributed in $[4, 7]$ and $[4, 9]$ in Figure 15(b) and 15(c) respectively, correspondingly, the task periods are in the range of $[16, 128]$ and $[16, 512]$ respectively. Again, the results obtained by this approach are only approximations of the real performances of the scheduling algorithms.

By these simulation experiments, we can see that for task sets with narrow task scale ranges, non-preemptive scheduling is potentially a better choice. As the range of task scales being enlarged the performance of non-preemptive scheduling degrades. This observation encourages us to study new scheduling policies such as grouping tasks with similar scales and then scheduling each group on a subset of the processors, which could be our future work.

10 Conclusions

As observed by Baruah [2], "global scheduling is fundamentally different from, and seems much more difficult than partitioned scheduling". In this paper, we take another stab at this tough problem by presenting new schedulability test conditions for work-conserving (necessarily global) non-preemptive scheduling on multiprocessor platforms, by building upon the techniques of Baker [5] and Baruah [2]. We firstly derive a linear-time test condition which works on any work-conserving non-preemptive scheduling algorithms. Then we improve the analysis and present test conditions of pseudo-polynomial time-complexity for EDF$_{np}$ and FP$_{np}$, which significantly outperform the existing result.

The pessimism of the analysis in this paper mainly comes from the assumption that worst-case interferences by all tasks happen simultaneously, which is actually not necessary. As future work, we plan to employ ILP or SAT methods to identify impossible scenarios, in order to obtain less-pessimistic schedulability tests.

References

[1] S. K. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Syst.*, 2006.

[2] S. K. Baruah, "Techniques for multiprocessor global schedulability analysis," in *RTSS*, 2007.

[3] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *RTSS*, 1991.

[4] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. 2004.

[5] T. P. Baker, "Multiprocessor edf and deadline monotonic schedulability analysis," in *RTSS*, 2003.

[6] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Syst.*, vol. 25.

[7] C. A. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal time-critical scheduling via resource augmentation," in *STOC*, 1997.

[8] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *RTSS*, 2001.

[9] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," in *ECRTS*, 2005.

[10] S. K. Baruah and T. P. Baker, "Schedulability analysis of global edf," in *Real Time Systems*, 2008.

[11] S. K. Baruah and T. P. Baker, "Global edf schedulability analysis of arbitrary sporadic task systems," in *ECRTS*, 2008.

[12] S. K. Baruah and N. Fisher, "Global fixed-priority scheduling of arbitrary-deadline sporadic task system," in *ICDCN*, 2008.

[13] B. Andersson and J. Jonsson, "Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling," *Technical Report 01-2, Chalmers University of Technology.*, 2001.

[14] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *RTSS*, 2007.

[15] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uni-processor scheduling," in *Technical Report, INRIA*, 1996.

[16] S. K. Baruah and S. Chakraborty, "Schedulability analysis of non-preemptive recurring real-time tasks," in *WPDRTS*, 2006.

[17] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan, "Time bounds for selection," *Journal of Computer and System Sciences*, 1973.

[18] T. P. Baker, "A comparison of global and partitioned edf schedulability tests for multiprocessors," in *Technical Report*, 2005.