Schedulability Analysis for Real-Time Systems with EDF Scheduling

Fengxiang Zhang and Alan Burns

Real-Time Systems Research Group, Department of Computer Science,

University of York, UK

{zhangfx, burns}@cs.york.ac.uk

Wednesday, 20th February, 2008

Abstract

Real-time scheduling is the theoretical basis of real-time systems engineering. Earliest Deadline First (EDF) is an optimal scheduling algorithm for uniprocessor real-time systems. The existing results on an exact schedulability test for EDF scheduling with arbitrary relative deadlines need to calculate the processor demand of the task set at every absolute deadline to check if there is an overflow in a specified time interval. The large amount of calculations restricts the use of EDF in practice. In this paper, we proposes new results on necessary and sufficient schedulability analysis for EDF scheduling; the new results reduce the calculation times in logarithm scales in all situations. For example a 16 tasks system that in the previous analysis had to check 858,331 points (deadlines) can, with the new analysis, be optimally check at just 12 points. The required calculation by the proposed analysis is stable for all kinds of task sets.

There is no restriction on the new results: each task can be periodic or sporadic, with relative deadline which can be less than, equal to or greater than its period; the tasks can have release jitter, and they can share non-preemptable resources in the system.

1. Introduction

Real-Time systems are playing a crucial role in our society, and in the last two decades, there is an explosive growth of real-time systems being used in our daily life and industry production, such as the chemical and nuclear plant control, space missions, flight control systems, military systems, telecommunications, multimedia systems, and so on. The most important attribute of real-time systems is that the correctness of such systems depends on not only the running results but also on the time at which results are produced. In other words, real-time systems have strict timing requirements that must be met. Real-time systems guarantee that all the timing requirements can be met by the theory of real-time scheduling and schedulability analysis.

In real-time scheduling theory, a real-time system comprises a set of real-time tasks, the tasks can be scheduled by a number of policies including fixed priority or dynamic priority algorithms. The success of a real-time system depends on whether all requests of the tasks can be guaranteed to complete their executions before their timing deadlines; if they can, then we say the task set is schedulable.

The most common dynamic priority scheduling policy for real-time systems is the earliest deadline first (EDF) algorithm which was introduced by Liu and Layland [10] in 1973. It has been proven by Dertouzos [7] to be optimal among all scheduling algorithms on a uniprocessor, in the sense that if a real-time task set cannot be scheduled by EDF, then this task set cannot be scheduled by any algorithm.

Liu and Layland [10] presented a necessary and sufficient schedulability condition for EDF scheduling under the assumption that all tasks' relative deadlines are equal to their periods (see Section 2 for definitions), the schedulability condition is that the total utilization of the task set is less than or equal to 1. However, in real-time systems, a task's relative deadline is not always equal to its period, so the above assumption severely restricts the usage of EDF in practice.

The existing results on exact schedulability analysis for EDF scheduling with arbitrary relative deadlines need to calculate the processor demand of the task set at every absolute deadline to check if there is an overflow in a specified time interval, this interval is bound by a certain value which guarantees we can find a failure point if the task set is not schedulable. In such an interval, there could be a very large number of absolute deadlines that need to be verified. The significant effort required to perform the exact schedulability test restricts the use of EDF in realistic systems, hence the EDF algorithm has not been

used as widely as the fixed priority algorithms in commercial real-time systems.

In this paper, we proposes new results on necessary and sufficient schedulability analysis for EDF scheduling, we refer it to as the Quick convergence Processor-demand Analysis (QPA) algorithm which builds on the traditional processor demand analysis. A tighter upper bound for the processor demand analysis is also presented. By intensive experiments, we show that QPA reduces the calculation times in logarithm scales in all situations, for example a 16 tasks system that in the previous analysis had to check 858,331 points (deadlines) can, with the proposed analysis, be optimally check at just 12 points. The required calculation by QPA is stable for all kinds of task sets.

We first prove that the traditional processor demand analysis can incorporate release jitter and blocking. We also show that the QPA algorithm can be integrated with both release jitter and blocking, hence general tasks with release jitter can share non-preemptable resources in our system model.

The rest of the paper is organized as follows. Section 2 describes the system model and notations used in this paper. Section 3 describes the existing results on exact schedulability tests based on the processor demand analysis for EDF scheduling with arbitrary relative deadlines. In Section 4, we present a tighter upper bound for the processor demand analysis.

In Section 5, we propose the QPA algorithm which provides fast schedulability tests for EDF scheduling, it is also necessary and sufficient. In Section 6, we give some examples to illustrate the proposed analysis. In Section 7, by intensive experiments on a large number of randomly generated task sets, we show that the QPA algorithm reduces the calculation times in logarithm scales in all situations, and the result is stable for all kinds of task sets.

Section 8 integrates the proposed new results with release jitter. In Section 9, we show that blocking and release jitter can be considered at the same time in the processor demand analysis, and that QPA can be integrate with both release jitter and blocking.

2. System Model

A hard real-time system comprises a set of n real-time tasks $\{\tau_1, \tau_2, ..., \tau_n\}$, each task consists of an infinite or finite stream of jobs or requests which must be completed before their deadlines. Let τ_i indicates any given task of the system. Each task can be periodic or sporadic.

Periodic tasks. All jobs of a periodic task τ_i have a regular interarrival time T_i, we call T_i the period of the periodic task τ_i. If a job for a periodic task τ_i arrives at time t, then the next job of task τ_i must arrive at $t + T_i$.

Sporadic tasks. The jobs of a sporadic task τ_i arrive irregularly, but they have a minimum interarrival time T_i, we call T_i the period of the sporadic task τ_i. If a job of a sporadic task τ_i arrives at t, then the next job of task τ_i can arrival at any time at or after $t+T_1$.

If there are periodic tasks in the system, since in realistic situations it is difficult to forecast or to handle the exact starting time of all tasks when a system starts up, the periodic tasks are supposed to start or arrive at the same time. Each job of task τ_i requires up to the same worst-case execution time which equals the task τ_i's worst-case execution time C_i, and each job of task τ_i has the same relative deadline which equals the task τ_i's relative deadline D_i. If a job of task τ_i arrives at time t, the required worst-case execution time C_i must be completed in D_i time units, and the absolute deadline of this job is $t+D_i$. Each task could have release jitter, when a job of task τ_i arrives at time t with the absolute deadline $t+D_i$, it will be released for execution at the latest time $t+J_i$ (the actual release time can be early than $t+J_i$).

At any time, an arrived job with a higher priority can preempt a lower priority job's execution. When a job completes its execution, the system chooses the pending job with the highest priority to execute. According to the EDF algorithm, the released job with the earliest absolute deadline is assigned the highest priority.

The following notation is used throughout the paper:

C_i —the worst-case execution time of task τ_i

D_i —the relative deadline of task τ_i

T_i —the period of task τ_i

n —the number of tasks in the system or the task set

U_i —the utilization of task τ_i, and $U_i = C_i / T_i$.

U —the total utilization of the task set, and $U = \sum_{i=1}^{n} C_i / T_i$

J_i —the maximum release jitter of task τ_i

r_i —a job (or a request) of task τ_i

3. Previous Results on Exact Schedulability Analysis

This section describes the previous research results on exact schedulability analysis for EDF scheduling with arbitrary relative deadlines. In 1980, Leung and Merrill [9] noted that a set of periodic tasks is schedulable if and only if all absolute deadlines in the period $[0, \max\{s_i\} + 2H]$ are met, where s_i is the start time of task τ_i, and $\min\{s_i\} = 0$. In 1990, Baruah et al. [2, 3] extended this condition for sporadic tasks system, and showed the task

set is schedulable if and only if: $\forall t > 0$, $h(t) \le t$, where $h(t)$ is the processor demand function given by:

$$h(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\} C_i \qquad (1)$$

Baruah et al. ([2, 3, 4]) showed that using the above necessary and sufficient schedulability tests, the value of t can be bound by a certain value.

Theorem 1 ([2, 3, 4]) A general task set (T_i and D_i are not related) is schedulable if and only if $U \le 1$ and

$$\forall t < L_a^1, \quad h_R(t) \le t$$

where L_a^1 is defined as follows:

$$L_a^1 = \max\{D_1, ..., D_n, \max_{1 \le i \le n}\{T_i - D_i\} \frac{U}{1-U}\} \qquad (2)$$

In 1996, Ripoll et al. [12] gave a tighter upper bound for the schedulability test under the assumption that all $D_i \le T_i$, the upper bound is:

$$L_a^2 = \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1-U}$$

Obviously, $$\frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1-U} \le \max_{1 \le i \le n}\{T_i - D_i\} \frac{U}{1-U}$$

However for the general task set in which D_i could be greater than T_i, the maximum relative deadline $\max_{1 \le i \le n}\{D_i\}$ should be reconsidered. Therefore the necessary and sufficient condition for schedulability becomes:

Theorem 2 ([4, 12]) A general task set is schedulable if and only if $U \le 1$ and

$$\forall t < L_a, \quad h(t) \le t,$$

where L_a is defined as:

$$L_a = \max\{D_1, ..., D_n, \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1-U}\} \qquad (3)$$

In 1996, Spuri [14] and Ripoll et al. [12] derived another upper bound for the time interval which guarantees we can find an overflow if the task set is not schedulable. This

interval is called the synchronous busy period (the length of the first processor busy period of the synchronous arrival pattern). However Ripoll et al. [12] only considered the situation of $D_i \le T_i$.

Definition 1 ([12, 14]) A synchronous busy period is a processor busy period in which all tasks are released simultaneously at the beginning of the processor busy period and then at their maximum rate, and ended by the first processor idle period (the length of such a period can be zero).

The length of the synchronous busy period L_b can be computed by the following process [12, 14]:

$$w^0 = \sum_{i=1}^{n} C_i , \tag{4}$$

$$w^{m+1} = \sum_{i=1}^{n} \left\lceil \frac{w^m}{T_i} \right\rceil , \tag{5}$$

the recurrence stops when $w^{m+1} = w^m$, and then $L_b = w^{m+1}$.

Lemma 1 ([14]) The length of the synchronous busy period is the maximum length of any possible busy processor period in any schedule.

Lemma 2 ([10]) When the EDF algorithm is used to schedule a set of tasks on a processor, there is no processor idle time prior to an overflow (deadline miss).

Theorem 3 ([14]) A general task set is schedulable if and only if $U \le 1$ and

$$\forall t \le L_b , \quad h(t) \le t ,$$

where L_b is the length of the synchronous busy period of the task set.

Lemma 3 $h(L_b) \le L_b$.

Proof. Let all tasks be released simultaneously at $t = 0$ and then at their maximum rate, according to Definition 1, the processor is always busy during $[0, L_b)$. Suppose $h(L_b) > L_b$, then the processor continues busy at and after $t = L_b$, so the busy period can be longer than L_b, this contradicts Lemma 1, hence $h(L_b) \le L_b$. □

Since there is no direct relationship between L_a and L_b, the time interval that needs to

be checked can be bound to the value $\min(L_a, L_b)$. As the processor demand $h(t)$ could only be changed at the absolute deadlines, the schedulability test becomes:

Theorem 4 ([4, 12, 14]) A task set is schedulable if and only if $U \le 1$ and

$$\forall t \in P,\ \ h(t) \le t,$$

where $$P = \{d_k \mid d_k = kT_i + D_i \wedge d_k < \min(L_a, L_b), k \in N\}.$$

4. Improvement to the Upper Bound L_a

In this section, we present a tighter upper bound for the schedulability test, there is no restriction on task parameters when we use the new upper bound.

Theorem 5 A general task set is schedulable if and only if $U \le 1$ and

$$\forall t \in P,\ \ h_R(t) \le t,$$

where $$P = \{d_k \mid d_k = kT_i + D_i \wedge d_k < L_a^*, k \in N\},$$

and where $$L_a^* = \max\{(D_1 - T_1), ..., (D_n - T_n), \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}\}. \quad (6)$$

Proof. When $t \ge \max_{1 \le i \le n}\{D_i - T_i\}$ $\Leftrightarrow$ $t \ge D_i - T_i$ $\Leftrightarrow$ $t - D_i \ge -T_i$ $\Leftrightarrow$ $\left\lfloor \frac{t - D_i}{T_i} \right\rfloor \ge -1$

$\Leftrightarrow$ $1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \ge 0$,

then we have: $h_R(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\} C_i = \sum_{i=1}^{n} (1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor) C_i$

$$= t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i)$$

If $U \le 1$ and the task set is not schedulable, $h_R(t) > t$

$\Leftrightarrow$ $$t < t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i)$$

$\Leftrightarrow$ $$t(1 - \sum_{i=1}^{n} \frac{C_i}{T_i}) < \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i)$$

$$\Leftrightarrow \qquad t < \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U} \qquad \square$$

Example of improvement

Here we give an example of a task set comprised of 5 tasks:

Task	Execution Time	Relative Deadline	Period
τ_1	3	4	9
τ_2	2	6	4
τ_3	4	2800	3100
τ_4	8	10170	10200
τ_5	3	406	430

The old result (see equation (3)): $L_a = 10170$, there are 3401 absolute deadlines in the interval $(0, L_a)$ that need to be checked.

The new result (see equation (6)): $L_a^* = 7.9$, there are only 2 absolute deadlines in the interval $(0, L_a^*)$ that need to be checked.

5. New Algorithm for Schedulability Analysis

The existing results on exact schedulability test for EDF scheduling need to check all the absolute deadlines in the time interval $(0, \min\{L_a, L_b\})$ when the task set is schedulable, this interval is bounded by a certain value which guarantees we can find a failure deadline if the task set is not schedulable. In a given interval, there can be a very large number of absolute deadlines needing to be checked.

In this section, we propose the Quick convergence Processor-demand Analysis (QPA) algorithm which provides fast and simple schedulability tests for EDF, it is also necessary and sufficient. By the proposed algorithm, we do not check every deadline, and we do not need all the values of deadlines in the interval even when the task set is schedulable.

We define L to be the minimum value of L_a^* and L_b. Considering that the upper bound L_a^* does not work (divide by 0) when the utilization of the task U is equal to 1, let L be defined as:

$$L = \begin{cases} \min(L_a^*, L_b) & U < 1 \\ L_b & U = 1 \end{cases} \qquad (7)$$

In this section, let d_i to be an absolute deadline of a job for task τ_i, and $d_i = kT_i + D_i$, $k \in N$. When a system is unschedulable, define d^{Δ} to be the largest d_i satisfying $0 < d_i < L \wedge h(d_i) > d_i$.

QPA words by starting with a value of t close to L and then iterating through a simple expression toward 0. The value of this t sequence converges for a unschedulable system to d^{Δ}, and converges for a schedulable system to 0 (or at $h(t) \le \min\{D_i\}$).

Lemma 4 For a unschedulable system, let $d_m = \max\{d_i \mid d_i < L\}$. If $h(d_m) \le d_m$, then $d^{\Delta} < h(d^{\Delta}) \le d'$, where $d' = \min\{d_i \mid d_i > d^{\Delta}\}$.

Proof. Since $d_m = \max\{d_i \mid d_i < L\}$, and $h(d_m) \le d_m$, we have $d^{\Delta} < d_m$, and $d' \le d_m < L$. Suppose $h(d^{\Delta}) > d'$, as $h(t)$ is a non-decreasing function with t, and $d' > d^{\Delta}$, we have $h(d') \ge h(d^{\Delta}) > d'$. This contradicts the condition that d^{Δ} is the largest d_i satisfying $0 < d_i < L \wedge h(d_i) > d_i$, so $d^{\Delta} < h(d^{\Delta}) \le d'$.

Lemma 5 For a unschedulable system, let $d_m = \max\{d_i \mid d_i < L\}$. If $h(d_m) \le d_m$, when $t \in [h(d^{\Delta}), L)$, then $d^{\Delta} < h(t) \le t$.

Proof. The period $[h(d^{\Delta}), L)$ can be divided into three intervals:

1) $t \in [h(d^{\Delta}), d')$, where $d' = \min\{d_i \mid d_i > d^{\Delta}\}$, from Lemma 4, $d^{\Delta} < h(d^{\Delta}) \le d'$, therefore $h(t) = h(d^{\Delta}) \le t$.

2) $t \in [d', d_m)$, then $h(t) = h(d_j)$, where $d_j = \max\{d_i \mid d_i \le t\}$. Suppose $t < h(t)$, we have $d_j \le t < h(t) = h(d_j)$, since $L_0 < d' \le d_j \le L$, this contradicts the condition that $d^{\Delta} = \max\{d_i \mid 0 < d_i < L \wedge h(d_i) > d_i\}$, therefore $h(t) \le t$.

3) $t \in [d_m, L)$, $h(t) = h(d_m) \le d_m \le t$.

Since $h(t)$ is a non-decreasing function with t, $t \ge h(d^{\Delta}) > d^{\Delta} \Rightarrow h(t) \ge h(d^{\Delta}) > d^{\Delta}$, hence we have $d^{\Delta} < h(t) \le t$ in each interval. □

Theorem 6 Let $E = \{d_i \mid d_i = kT_i + D_i, d_i < L, k \ge 0\} = \{d_1, d_2, ..., d_m\} \ne \varnothing$, where $d_1 < d_2 < ... < d_m$. If $U \le 1$ and the task set is not schedulable, then there exists one or more absolute deadlines $d_i \in E \wedge h(d_i) > d_i$. Let d^{Δ} be the largest of such d_i, we can use the following algorithm to find d^{Δ}. If the following iterative process could not find d^{Δ}, then the task set is schedulable.

$t = d_m$;

while($h(t) \le t \wedge h(t) > d_1$)

{***if*** ($h(t) < t$) $t = h(t)$;

else $t = d_i$; where $d_i < t \le d_{i+1}$

}

if ($h(t) \le d_1$) the task set is schedulable;

else the task set is not schedulable (then t equals d^{Δ});

Proof. Suppose the task set is not schedulable.

If $h(d_m) > d_m$, the iteration stops as $h(t) > t$.

If $h(d_m) \le d_m$, we have $d_m > d^{\Delta}$, then there are three cases during the iterative process:

Case 1. $h(t) < t$: at the beginning of the iteration $t = d_m > d^{\Delta}$, since $h(t)$ is a non-decreasing function with t, $h(t) \ge h(d^{\Delta})$, then we have $h(d^{\Delta}) \le h(t) < t < L$, from Lemma 5, $h(d_m) \le d_m \wedge t \in [h(d^{\Delta}), L] \Rightarrow d^{\Delta} < h(t) \le t$, so after $t \leftarrow h(t)$, we still have $d^{\Delta} < h(L_0) \le h(t) < t < L$. Therefore t is always greater than d^{Δ} in this case.

Case 2. $h(t) = t$: at this time, t is still larger than d^{Δ}, if we let $t \leftarrow d_i$, where $d_i < t \le d_{i+1}$, obviously $t = d_i \ge d^{\Delta}$. If $t = d_i = d^{\Delta}$, then $h(t) > t$, the iterative process stops. If $t = d_i > d^{\Delta}$, from Lemma 4, $d^{\Delta} < h(d^{\Delta}) \le d'$ where $d' = \min\{d_i \mid d_i > d^{\Delta}\}$, so we have $h(d^{\Delta}) \le d' \le t < L$, from Lemma 5, the process enters Case 1 or Case 2 again.

Case 3. $t = h(d^{\Delta})$: from Lemma 4, $d^{\Delta} < h(d^{\Delta}) \le d'$, so $h(t) = h(d^{\Delta}) = t$, that is a situation of Case 2, at this time, if we let $t = d_i$ such that $d_i < t \le d_{i+1}$, then $t = d^{\Delta}$, $h(t) > t$, and the recurrence stops.

From the above discussion, if the task set is not schedulable, then during the whole iterative process, the value of t is always greater than or equal to d^{Δ} (stopped with $t = d^{\Delta}$), and we have $h(t) \ge h(d^{\Delta}) > d^{\Delta} \ge d_1$. Therefore when $h(t) \le d_1$, the task set is schedulable. □

When $h(t) \le d_1$, if we change the stopping condition to let the iterative process continue, then after one or two more iteration, t will converge at 0. Theorem 6 is a necessary and sufficient condition for schedulability, it can be presented as: a general task set is schedulable if and only if $U \le 1$ and the algorithm of Theorem 6 could not find d^{Δ}.

The algorithm in Theorem 6 needs to calculate the value of all absolute deadlines in

$E = \{d_i \mid d_i = kT_i + D_i \wedge d_i < L, k > 0\}$, the calculation has the complex $O(m)$, where m is the number of deadlines in E. The algorithm also needs to eliminate the same copies of the deadlines in E and sort all deadlines in order, both of these two steps have the complex of $O(m^2)$ in the worst case. When m is very large, the workload of these calculations can be huge, but fortunately this is not necessary. We do not need to know the exact number of deadlines in E, and we do not even need to know all the values of deadlines in E. Theorem 6 can be changed to the following theorem.

Theorem 7 Let all tasks be released simultaneously at $t = 0$ then at their maximum rate, and d_i be an absolute deadline of a job arrives after $t = 0$, denote $d_{\min} = \min\{D_i\}$. If $U \le 1$, the task set is schedulable if and only if the result of the following algorithm is $h(t) \le d_{\min}$.

$t = \max\{d_i \mid d_i < L\}$;
while $(h(t) \le t \wedge h(t) > d_{\min})$
{***if*** $(h(t) < t)$ $t = h(t)$;
else $t = \max\{d_i \mid d_i < t\}$;
}
if $(h(t) \le d_{\min})$ the task set is schedulable;
else the task set is not schedulable;

Proof. $d_{\min} = \min\{d_i\}$ is equal to d_1 in Theorem 6, $\max\{d_i \mid d_i < L\}$ is equal to d_m in Theorem 6, and $\max\{d_i \mid d_i < t\}$ is equal to d_i where $d_i < t \le d_{i+1}$ in Theorem 6. □

In the iterative process of Theorem 7, t takes the value $h(t)$, when $h(t) < t$ progress towards zero is made. Only when $h(t) = t$ do we need to force the iterative process to take a value less than $h(t)$. This is when we need to compute $\max\{d_i \mid d_i < t\}$ to let the iteration continue, $\max\{d_i \mid d_i < t\}$ can be calculated by the following approach.

For a single task τ_j, the last arrived job before t is released at:

$$(\left\lceil \frac{t}{T_j} \right\rceil - 1)T_j,$$

the absolute deadline of this job is:

$$d_j = (\left\lceil \frac{t}{T_j} \right\rceil - 1)T_j + D_j. \tag{8}$$

If $D_j < T_j$, then d_j in equation (8) is $\max\{d_j \mid d_j \le t\}$. To consider the case of $D_j \ge T_j$, we can let d_j move on to the previous deadline $d_j = d_j - T_j$, until $d_j < t$.

For the task set, $\max\{d_i \mid d_i \le t\}$ is the largest such d_j for each task.

Let the initial value of $d^t_{\max} = 0$, the value of $\max\{d_i \mid d_i \le t\}$ can be obtained by:

---

for $(j = 1; j \le n; j{+}{+})$

$\{ d_j = (\lceil t / T_j \rceil - 1)T_j + D_j;$

while $(d_j > t)$ $d_j = d_j - T_j;$

if $(d_j > d^t_{\max})$ $d^t_{\max} = d_j;$

$\}$

After the recurrence, $d^t_{\max} = \max\{d_i \mid d_i \le t\}$.

The above algorithm for finding the $\max\{d_i \mid d_i \le t\}$ has the complex $O(n)$, the workload is only equivalent to calculating the value of $h(t)$ one time.

In our experiments, we observed that $h(t) = t$ occurred very rarely for schedulable task sets, and for unschedulable task sets, in most cases, $h(t) = t$ only occurred at the end of Theorem 7's iteration during the schedulability test.

6. Illustration Examples

In this section, first we give two examples to illustrate the analysis in Section 5, and then we give examples to illustrate the necessary of the formula $t = \max\{d_i \mid d_i < t\}$ in the algorithm of Theorem 7**.**

Example A

In this example, the task set includes 8 tasks:

Task	Execution Time	Relative Deadline	Period
τ_1	6000	18000	31000
τ_2	2000	9000	9800
τ_3	1000	12000	17000
τ_4	90	3000	4200
τ_5	8	78	96
τ_6	2	16	12

τ_7	10	120	280
τ_8	26	160	660

The schedulability is tested by the following steps:

Step 1. Calculate the utilization of the task set, $U \cong 0.803 \le 1$.

Step 2. Calculate upper bound L_a by equation (3), $L_a = 18000$.

(There are 1735 absolute deadlines in L_a.)

Step 3. Calculate upper bound L_b by equations (4)(5), $L_b = 16984$.

(There are 1638 absolute deadlines in L_b.)

(There are 1638 absolute deadlines that require to be checked by the old approach.)

Step 4. Calculate upper bound L_a^* by equation (6), $L_a^* = 15357$.

As $L_a^* < L_b$, $L = L_a^* = 15357$; $d_{\min}$ =16 and $\max\{d_i \mid d_i < L\}$ =15352.

Step 5. Verify the schedulability by the QPA algorithm given by Theorem 7:

1) $t = 15352$, $h(t)$=8282,
2) $t = 8282$, $h(t)$=2884,
3) $t = 2884$, $h(t)$=950,
4) $t = 950$, $h(t)$=318,
5) $t = 318$, $h(t)$=112,
6) $t = 112$, $h(t)$=26,
7) $t = 26$, $h(t)$=2;

Since $h(t) = 2 \le d_{\min}$, the task set is schedulable.

Only 7 iteration required compared to 1638 previously.

Note step 2 is not needed for the new results as L_a^* is always less than or equal to L_a.

Example B

In this example, the task set is taken from the evaluation in Section 7 (the parameters are generated automatically). The period range of the 16 tasks is 1-10^6, and their parameters are as follows:

Task	Execution Time	Relative Deadline	Period
τ_1	0.046017	1.419897	1.324642
τ_2	0.103668	1.819094	1.665016
τ_3	0.005133	0.341237	2.894035
τ_4	0.867954	17.024084	14.970213

τ_5	0.138762	37.186903	47.98906
τ_6	4.374349	42.335452	90.746475
τ_7	21.725475	287.330462	372.991594
τ_8	18.601216	673.81609	847.449375
τ_9	164.298311	1874.27352	1762.311388
τ_{10}	624.972457	5915.216558	5787.904261
τ_{11}	1641.81716	7625.959698	15549.16986
τ_{12}	1129.07592	4560.327895	46697.68032
τ_{13}	13712.6729	92575.63579	80125.00202
τ_{14}	12241.42	379004.064	439136.1428
τ_{15}	24268.6361	781900.4736	715880.6276
τ_{16}	48061.4305	939573.2658	1000000

The schedulability is tested by the following steps:

Step 1. Calculate the utilization of the task set, $U = 0.9 \le 1$.

Step 2. Calculate upper bound L_a by equation (3), L_a=939573.265758.
(There are 1695376 absolute deadlines in L_a.)

Step 3. Calculate upper bound L_b by equations (4)(5), L_b=475686.09375.
(There are 858331 absolute deadlines in L_b.)

(There are 858331 absolute deadlines that need to be verified by previous results.)

Step 4. Calculate upper bound L_a^* by equation (6), $L_a^* = 66019.846$.

As $L_a^* < L_b$, $L = L_a^* = 66019.846$; $d_{\min}$=0.341237 and $\max\{d_i \mid d_i < L\}$=66019.710586.

Step 5. Verify the schedulability by the QPA algorithm given by Theorem 7:

1. t=66019.710586, h(t)=40798.678690,
2. t=40798.678690, h(t)=25950.533926,
3. t=25950.533926, h(t)=16663.199224,
4. t=16663.199224, h(t)=10272.873244,
5. t=10272.873244, h(t)=7161.185345,
6. t=7161.185345, h(t)=4296.913363,
7. t=4296.913363, h(t)=1551.081489,
8. t=1551.081489, h(t)=445.414149,
9. t=445.414149, h(t)=113.948337,
10. t=113.948337, h(t)=21.893751,
11. t=21.893751, h(t)=2.992976,
12. t=2.992976, h(t)=0.200835;

Since $h(t) \le d_{\min}$, the task set is schedulable.

Again a significant reduction, 12 compared with 858331.

Necessary of the algorithm $t = \max\{d_i \mid d_i < t\}$

In the iteration of Theorem 7, when $h(t) = t$, we need to find d_i where $d_i = \max\{d_i \mid d_i < t\}$, and let $t = \max\{d_i \mid d_i < t\}$, to make the iteration continuing. Although the chance of $h(t) = t$ is very small for the task set which is schedulable, and for the unschedulable task set $h(t) = t$ only appears at the end of the iteration in most cases, judging if $h(t) = t$ and letting assigning $t = \max\{d_i \mid d_i < t\}$ is necessary for the QPA algorithm. Here we give two examples to illustrate this, one task set is schedulable and the other one is unschedulable.

Example 1.

Task	Execution Time	Relative Deadline	Period
τ_1	8	11	60
τ_2	12	20	170
τ_3	6	26	120
τ_4	7	80	110

For this task set, $L = 33$, and $d_{\min} = 11$, $d_{\max} = 26$. The iteration is shown as follows:

1. t=26, $h(t)$=26
2. t=20, $h(t)$=20
3. t=10, $h(t)$=8

Since $h(t) < d_{\min}$, the task set is schedulable.

Example 2.

Task	Execution Time	Relative Deadline	Period
τ_1	8	10	60
τ_2	12	19	170
τ_3	10	30	210
τ_4	6	36	190
τ_5	8	70	280
τ_6	7	90	320

For this task set, $L = 51$, and $d_{\min} = 10$, $d_{\max} = 36$. The iteration is shown as follows:

1. t=36, $h(t)$=36
2. t=30, $h(t)$=30
3. t=19, $h(t)$=20

Since the iteration stops with $h(t) > d_{\min}$, the task set is not schedulable.

7. Experiments and Evaluations

This section describes experiments that have been conducted to evaluate the performance of the Quick convergence Processor-demand Analysis (QPA) algorithm which is proposed in Section 5, and we compare the number of calculation times needed for the upper bounds L_a, L_b, L_a^*, and the QPA algorithm by intensive experiments on a large number of task sets with randomly generated parameters.

Utilizations generation policy.

In order to get a uniform distributed task utilizations in the range 0-1, we use the UUniFast algorithm [6] to generate the task utilizations. Bini and Buttazzo [6] showed that the UUniFast algorithm can efficiently generate task utilizations with uniform distributions.

Periods generation policy.

The task periods are generated according to an exponent distribution. Let $T_{\max} = \max_{1 \le i \le n}\{T_i\}$, and $T_{\min} = \min_{1 \le i \le n}\{T_i\}$. In order to make sure the periods are uniformly distributed in the given range (the maximum value of $T_{\max} / T_{\min}$), the range of the periods are divided into the intervals $e^o \sim e^1$, $e^1 \sim e^2$, $e^2 \sim e^3$, ..., if there are k intervals, then $\lfloor (n-1)/k \rfloor$ task periods are generated randomly in each interval, and one of the rest ($(n-1) \bmod k$) task periods is generated randomly from each intervals.

For example, if the task number is 14, and the given periods range is 1-100, first let $T_{14} = 100$, since $\ln 100 \cong 4.605$, the range is divided into 5 intervals, they are $e^o \sim e^1$, $e^1 \sim e^2$, …, $e^4 \sim e^{4.605}$, then 2 task periods are generated randomly in each interval, and for the rest 3 periods, one is generated in each interval, shown as following:

$e^o \sim e^1$: τ_1, τ_2, τ_{11}

$e^1 \sim e^2$: τ_3, τ_4, τ_{12}

$e^2 \sim e^3$: τ_5, τ_6, τ_{13}

$e^3 \sim e^4$: τ_7, τ_8,

$e^4 \sim e^{4.605}$: τ_9, τ_{10},

Let T_n be the maximum value of $T_{\max} / T_{\min}$. When $\ln T_n - \lfloor \ln T_n \rfloor \le 0.1$, such as

$T_n = e^{6.02}$, there is no necessary to have an interval $e^6 \sim e^{6.02}$, then we let the last interval be $e^5 \sim e^{6.02}$.

Relative deadlines generation policy.

The relative deadline of each task D_i is generated randomly from $[a,b]$, where a is the lower bound value of D_i, and b is the upper bound value of D_i. In our default generation policy, the value of each a: when $C_i < 10$, $a = C_i$; when $10 \le C_i < 100$, $a = 2 \times C_i$; when $100 \le C_i < 1000$, $a = 3 \times C_i$; when $C_i \ge 1000$, $a = 4 \times C_i$. The default value $b = 1.2 \times T_i$.

In some experiments, we may change the default value of a and b for the purpose of observation. If we do not specify, the relative deadlines are generated by the above default policy.

On the graphs, when the y axis is labeled "Number of $h(t)$ Calculations", for the old methods, it means how many absolute deadlines have to be verified in each upper bound when the task sets are schedulable, or how many deadlines have been verified before the end of the schedulability test when the task sets are unschedulable (verifying one deadline requires calculate the function $h(t)$ one time). For the QPA algorithm, this label just presents the required calculation for the proposed schedulability test.

A reasonable metric to compare results is to measure the number of times the processor demand function $h(t)$ has to be calculated. Since the old results need to check all the absolute deadlines in the upper bound when a task set is schedulable, we do the experiments separately for the schedulable and the unschedulable task sets, i.e. when we experiment on schedulable task sets, if a generated task set is unschedulable, then the program discards it and does not count it into the experimental results. However all experiments use the same default generation policy described above (unless an alternative is specified).

7.1 Experiments on Schedulable Task sets

In this section, we experiment on the task sets which are schedulable. All task sets are randomly generated according to the above default policies or the given policy if it is specified. If a generated task set is schedulable, then the program counts it into the experimental results; and if a task set is unschedulable, the program does not count it into the results. Except experiment (G), each point on the result diagrams are the average of

2,000 randomly generated task sets which are schedulable.

Due to the magnitude of the improvement, all the comparison graphs use logarithm scales on the y-axis. In all experiments, 4 situations are compared, on the graphs, "L_a", "L_b", and "L_a^*" present the number of deadlines that need be checked in L_a, L_b, and L_a^* be the old methods, "QPA algorithm" presents the required calculation by QPA. Note all tests are necessary and sufficient and hence no task sets pass one test while failing the others.

As the density of a task set $\Delta = \sum_{i=1}^{n} C_i / \min\{D_i, T_i\} \le 1$ is a sufficient schedulability condition for EDF scheduling [11], in the experiments, we also test the density of each schedulable task set, to see how much percentage of the task sets have $\Delta > 1$; in which case we cannot judge the schedulability by this simple but sufficient test.

(A) Impact of the number of tasks

In this experiment, we let each task set's utilization be 0.9, the maximum value of $T_{\max} / T_{\min}$ is 10000, then the number of the required calculation times is a function of the number of tasks.

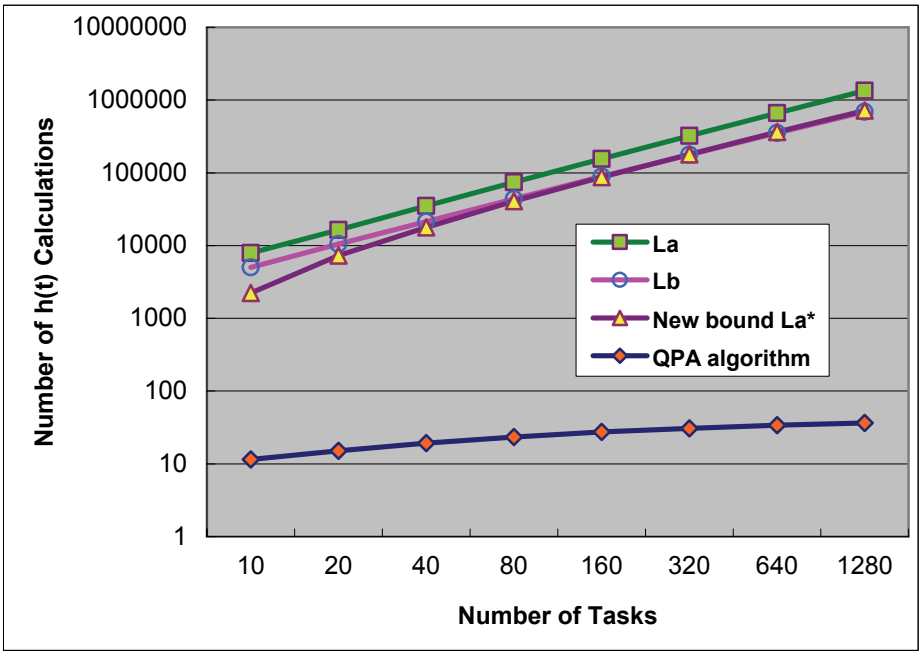

Figure 1. Impact of the number of tasks

The percentage of the task sets with density larger than 1 in this experiment:

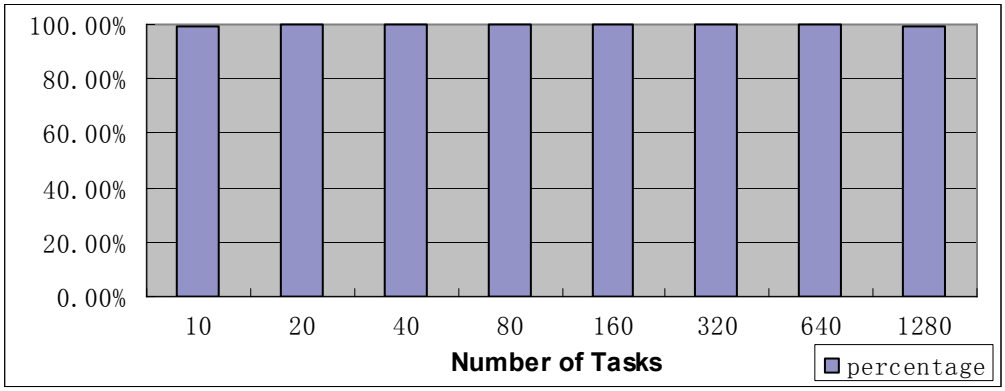

Figure 2. Percentage of the (schedulable) task sets with density larger than 1

From Figure 2, we can see that nearly all the schedulable task sets in the experiment have the density $\Delta > 1$, so that nearly all the task sets require QPA to test their schedulability.

(B) Impact of the task periods range

In this experiment, we let the number of tasks for each task set be 30, and each task set's utilization be 0.9, then the number of calculation times is a function of the maximum value of $T_{\max} / T_{\min}$.

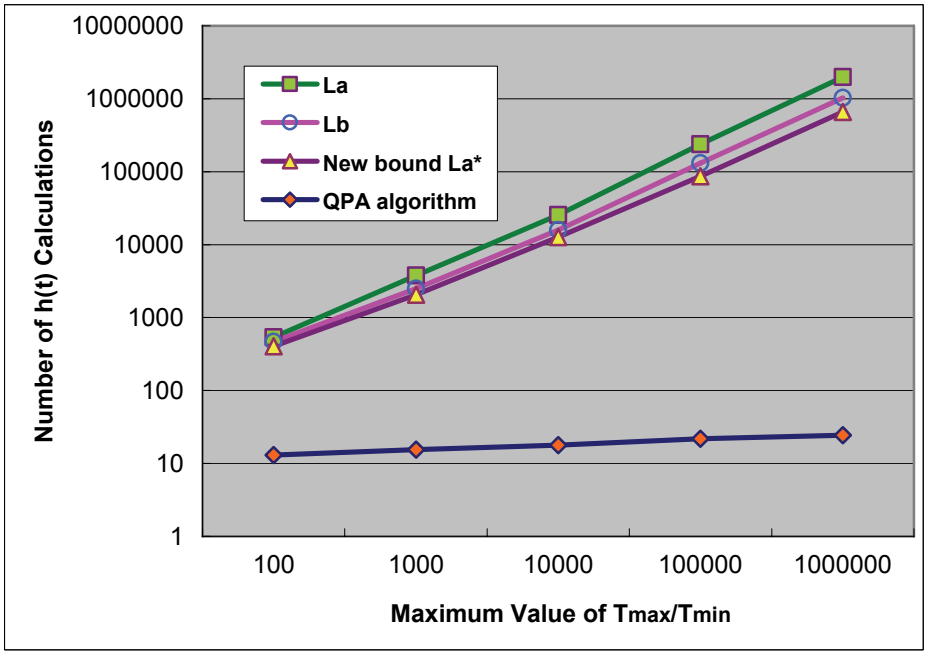

Figure 3. Impact of the task periods range

The percentage of the task sets with density larger than 1:

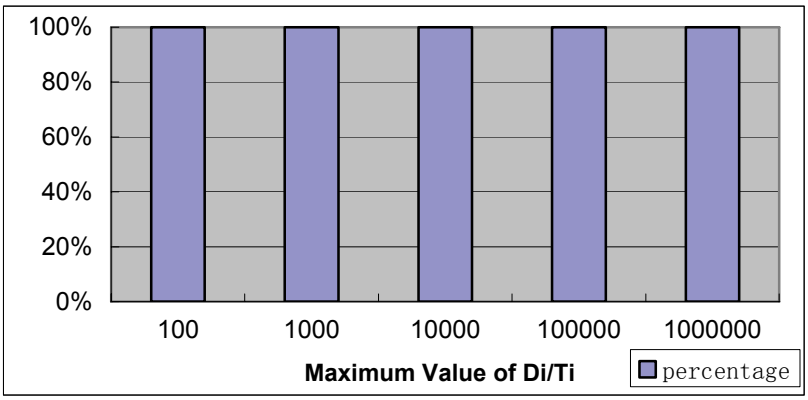

Figure 4. Percentage of the (schedulable) task sets with density larger than 1

(C) Impact of the utilization

In this experiment, we let the size of each task set be 30, and $T_{\max} / T_{\min} \leq 10000$, then the number of calculation times is a function of the task set's utilization.

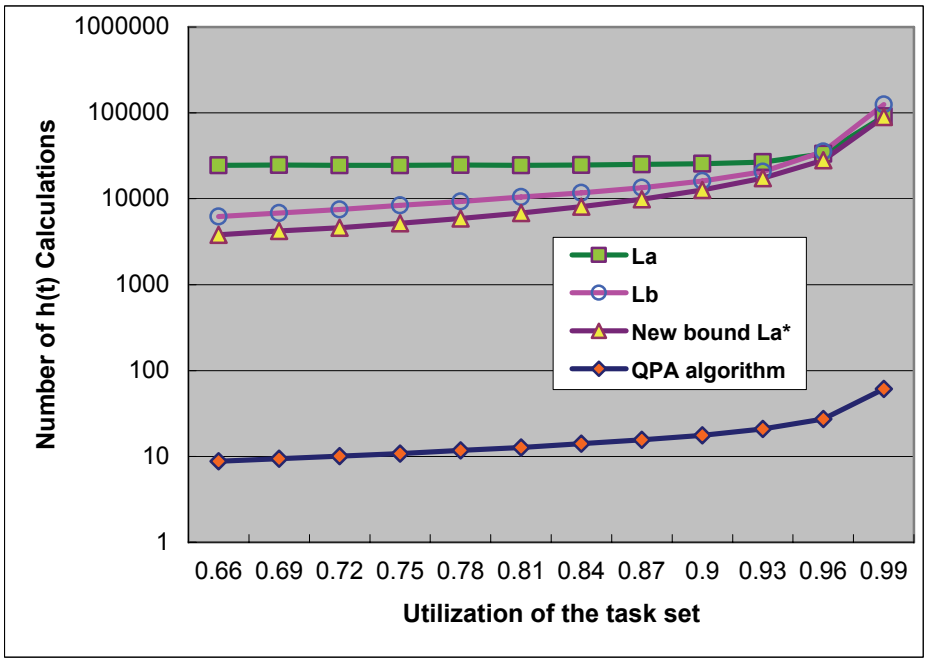

Figure 5. Impact of the task set's utilization

The percentage of the task sets with density larger than 1:

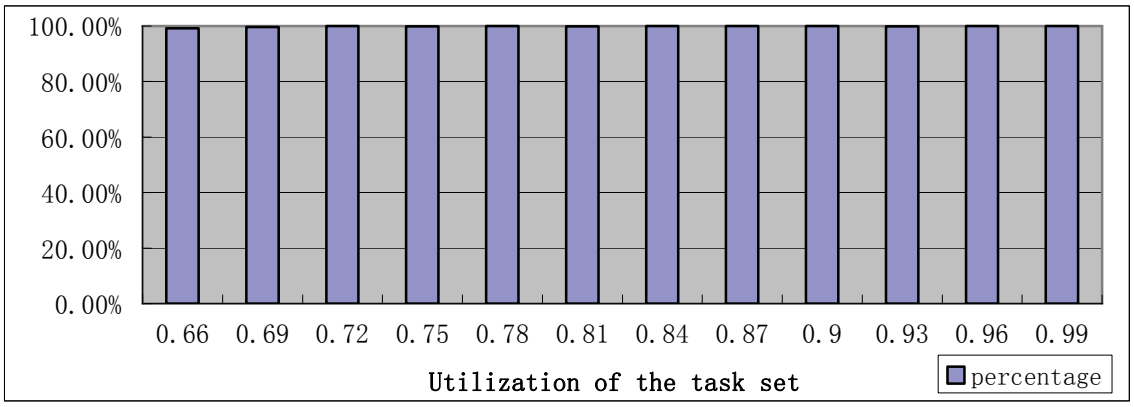

Figure 6. Percentage of the (schedulable) task sets with density larger than 1

(D) Impact of the maximum value D_i / T_i

We change the default generation policy of relative deadlines, let each relative deadline be generated randomly from a to b, where a remains the same value as the default generation policy, and $b = \max\{Di / Ti\} \times T_i$. The utilization of each task set is 0.9, task number is 30, and $T_{\max} / T_{\min} \le 10000$. Then the number of calculation times is a function of the value $\max\{Di / Ti\}$.

Experiment 1. Let b be changed from $1.1 \times T_i$ to $2.2 \times T_i$.

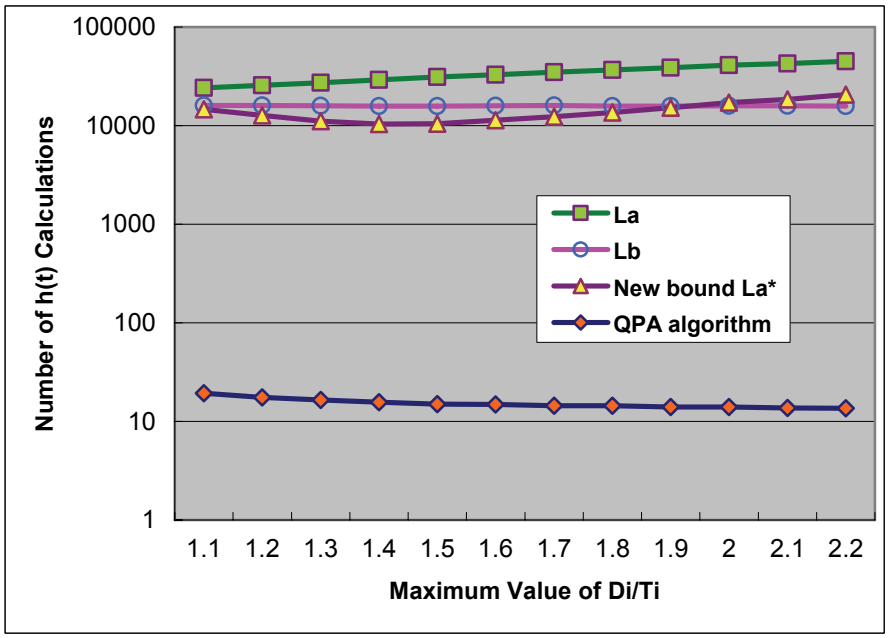

Figure 7. Impact of the maximum value of Di/Ti

The percentage of the task sets with density larger than 1:

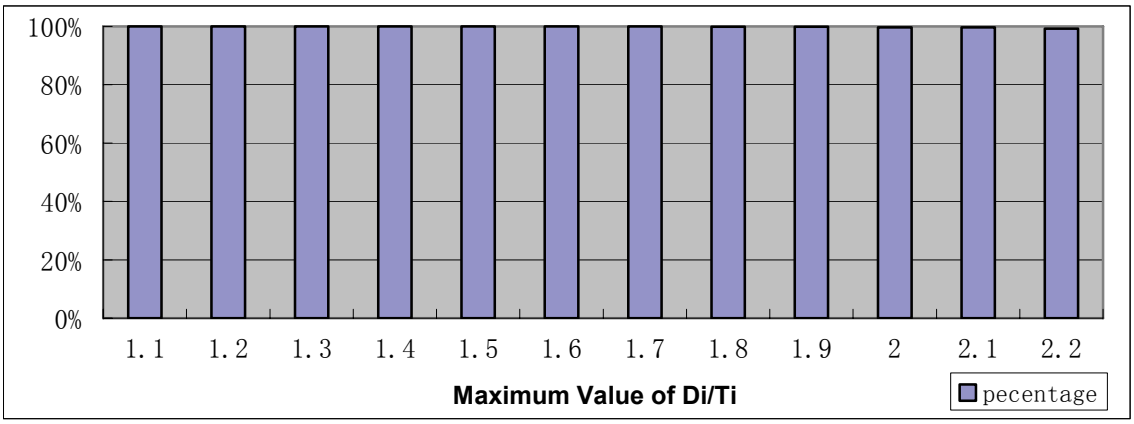

Figure 8. Percentage of the (schedulable) task sets with density larger than 1

Experiment 2. Let b be changed from $1.2\times T_i$ to $3.2\times T_i$.

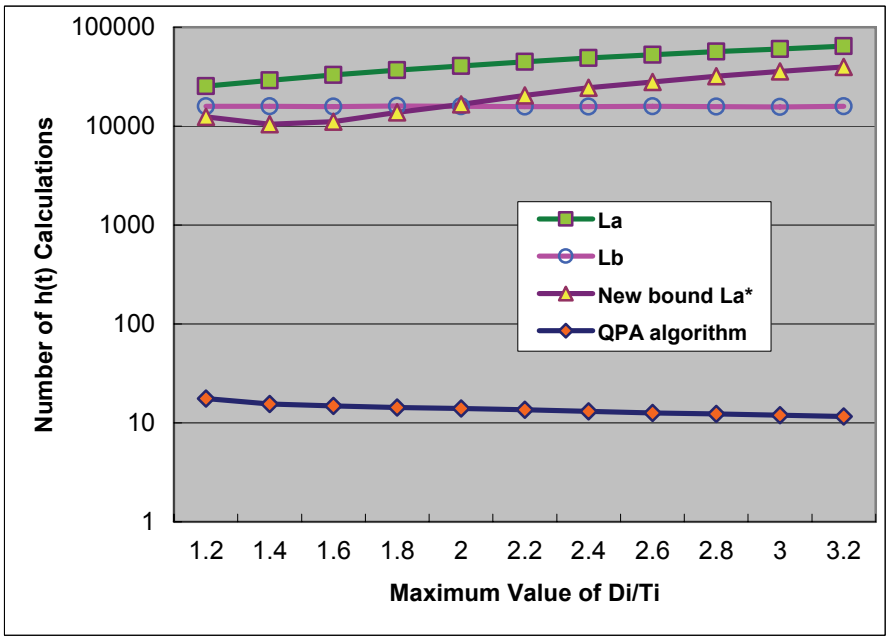

Figure 9. Impact of the maximum value of Di/Ti

The percentage of the task sets with density larger than 1:

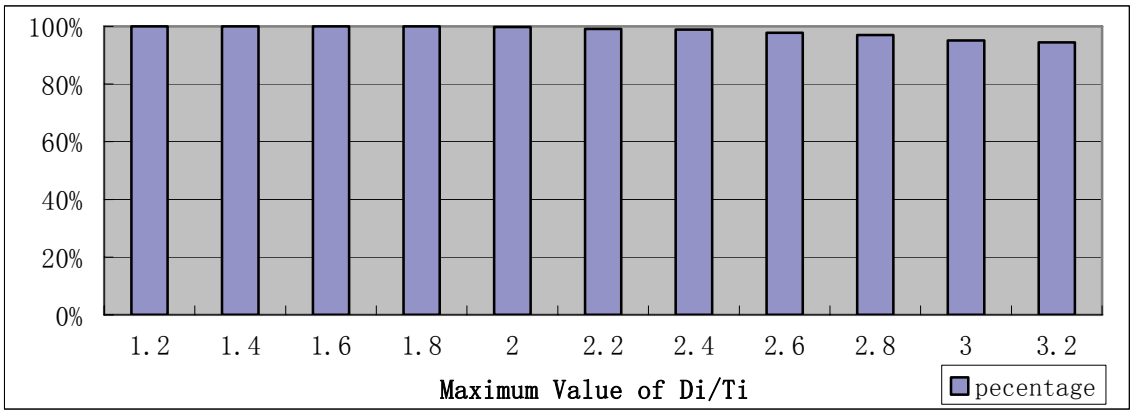

Figure 10. Percentage of the (schedulable) task sets with density larger than 1

Our default generation policy for each relative deadline is $b = \max\{D_i / T_i\} = 1.2$. From the above two experiments, we can see that the QPA algorithm can perform faster if we let $b > 1.2$. When $\max\{D_i / T_i\} \le 2.2$, the percentage of the schedulable task sets with the density larger than one is nearly 100%.

(E) Impact of the minimum value D_i / T_i

In this experiment, in order to control the maximum distance from each D_i to T_i, we let each D_i be generated randomly from a to b, where $a = \min\{Di / Ti\} \times T_i$, and b remains the same value of the default generation policy, that is $b = 1.2 \times T_i$.

We let the utilization of each task set be 0.9, the size of each task set be 30, and $T_{\max} / T_{\min} \le 10000$, then the calculations is a function of $\min\{Di / Ti\}$.

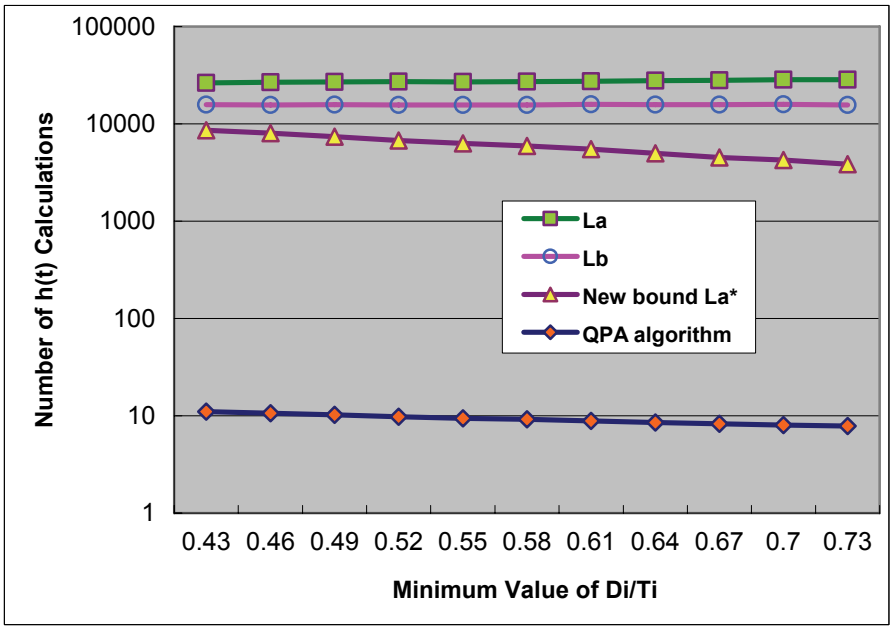

Figure 11. Impact of the minimum value Di/Ti

The percentage of the task sets with density larger than 1:

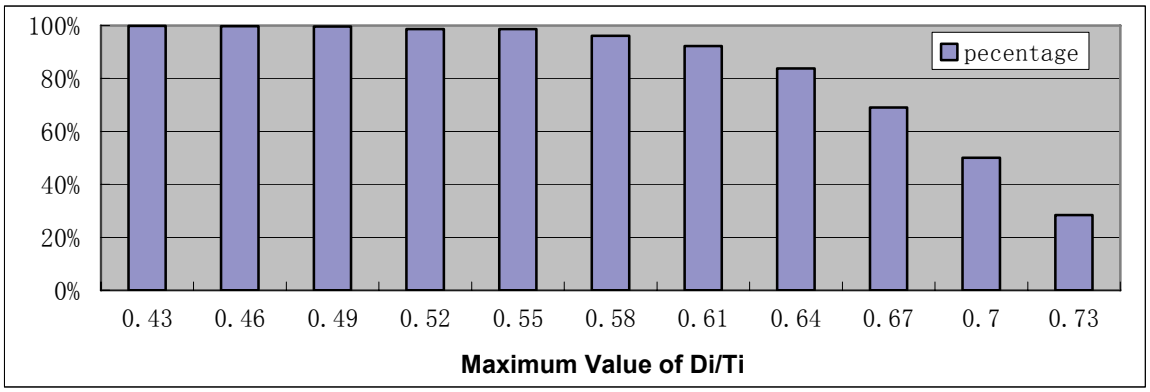

Figure 12. Percentage of the (schedulable) task sets with density larger than 1

(F) Fix the value of D_i / T_i

In this experiment, in order to observer the performance of QPA when all $D_i \le T_i$, and the impact of the distance from each D_i to T_i, we fix the value of each D_i / T_i, let the number of calculation times be a function of the fixed value D_i / T_i.

When all $D_i \le T_i$, the upper bound $La*$ becomes:

$$La* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U},$$

that is equal to L_a^2 which is given by Ripoll et al. [12] under the assumption all $D_i \le T_i$.

Let $k = D_i / T_i$, $k \le 1$, then the density of each task set is:

$$\Delta = \sum_{i=1}^{n} \frac{C_i}{\min\{D_i, T_i\}} = \sum_{i=1}^{n} \frac{C_i}{D_i} = \sum_{i=1}^{n} \frac{C_i}{kT_i} = \frac{1}{k}\sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{1}{k}U,$$

If we want $\Delta > 1$, $U / k > 1 \Rightarrow k < U$.

We let the utilization of each task set be 0.9, tasks number be 30, and the maximum value of $T_{\max} / T_{\min}$ is 10000, so $k < 0.9$.

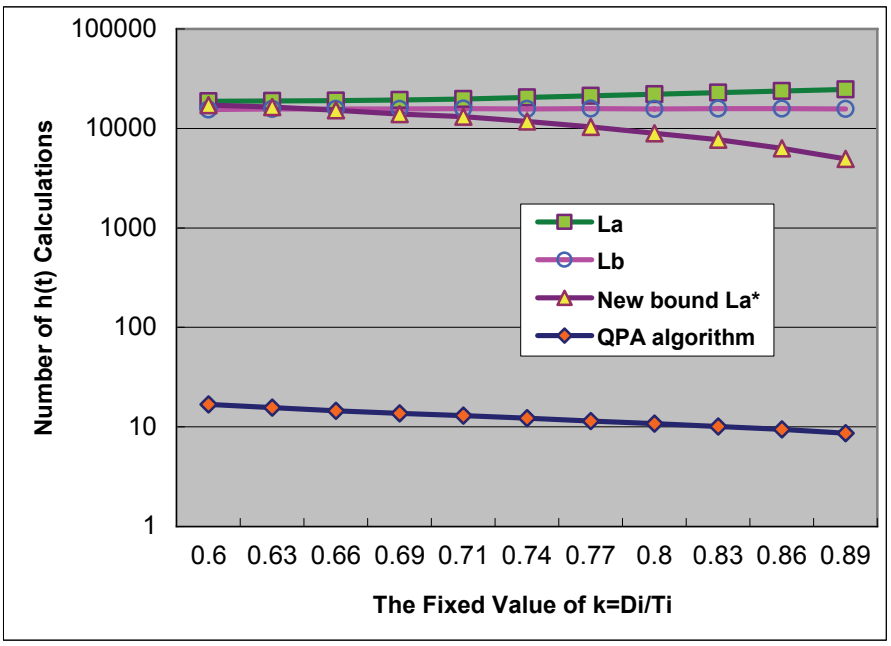

Figure 13. Impact of the fixed value of Di/Ti

Although we only experiment on the schedulable task sets, we counted the number of the unschedulable task sets, denote it to be N_{unsche}, so at each point on the graph, the percentage of the schedulable task sets is:

$$\frac{2000}{(2000+N_{unsche})}\times 100\%$$

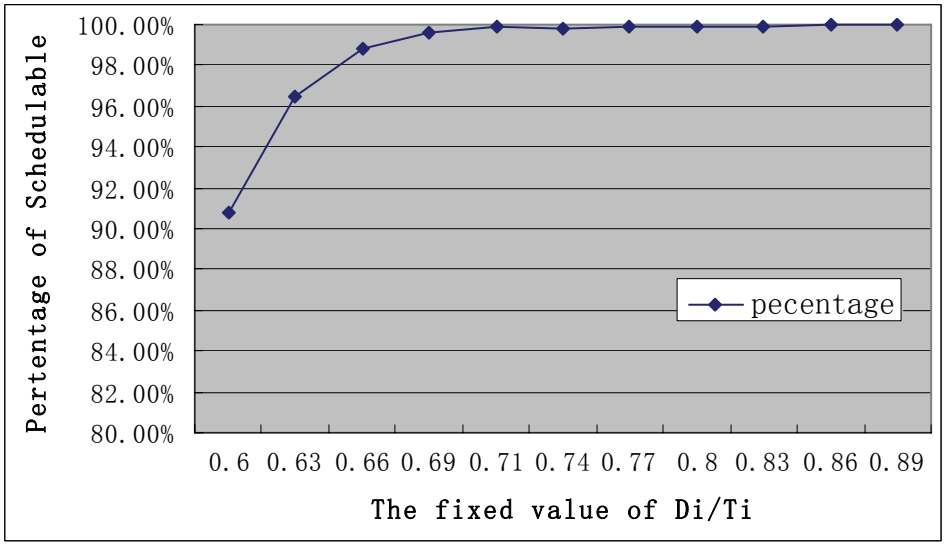

Figure 14. Percentage of the schedulable task sets

(G) Frequency distribution of the task sets

This experiment explores how many tested task sets can complete their schedulability tests in a given number of $h(t)$ calculations. We divide each interval on the x axis into 10;

this means if a schedulable task set needs 126 calculations to complete its schedulability test, then this task set is counted into the interval 120~130 on the x axis. The value of the y axis presents the counted number of the task sets.

The experiment tests 80,000 randomly generated task sets which are schedulable, for each task set, the utilization is 0.9, the tasks number is 30, and the maximum value of $T_{\max} / T_{\min}$ is 10000.

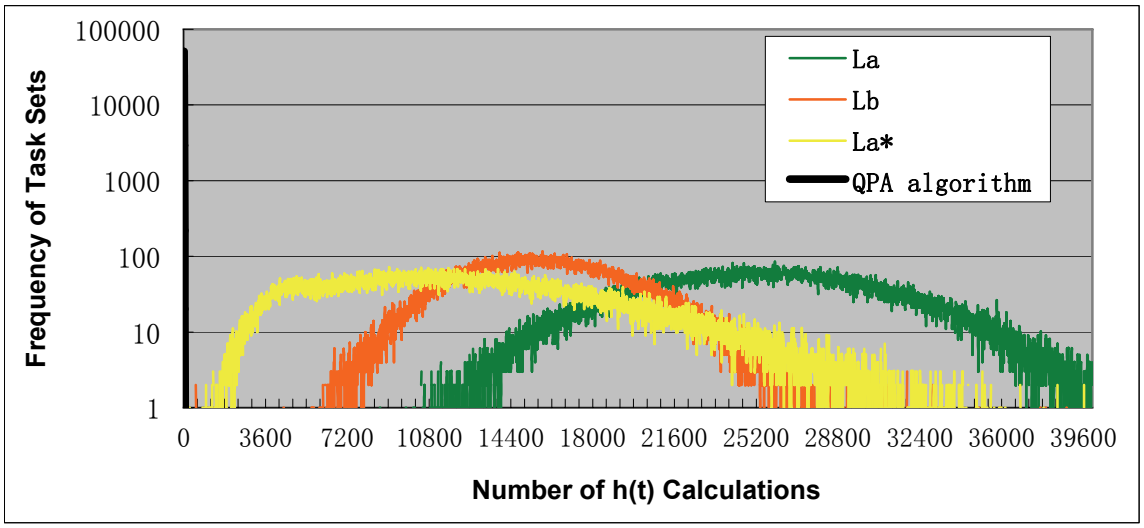

Figure 15. Frequency distribution experiment based on 80,000 schedulable task sets

Since the numbers of calculation times needed by the QPA algorithm for all task sets are very close to the origin of the coordinates, in Figure 15, the line of the algorithm is superpose on the y axis. In order to see clearly the frequency distribution for the algorithm, we give the following graph according to the above experiment.

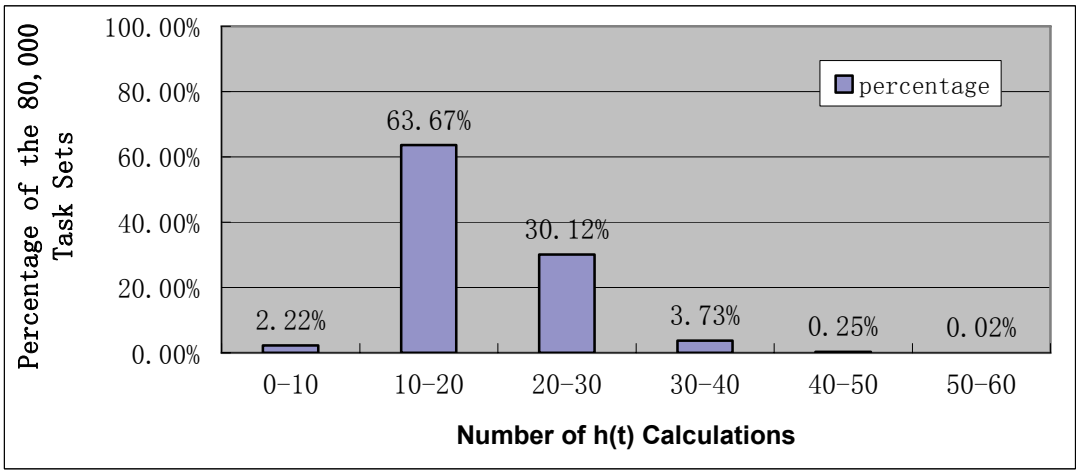

Figure 16. Frequency distribution of QPA

We can see from Figure 16 that most task sets complete each schedulability test in less than 30 times calculations of $h(t)$, and all the 80,000 task sets in our experiment complete each schedulability test in less than 60 times calculations.

7.2 Experiments on Unschedulable Task sets

This section describes experiments on unschedulable task sets. All task sets are generated according to the default generation policies if not specify, and only the unschedulable task sets are counted into the experimental results.

It is difficult to decide how to compare with the previous results on exact schedulability tests, since no literature mentions or explores what order should be used to check all the absolute deadlines by previous research results. For a schedulable system all deadlines need to be checked and hence the order is immaterial, but for a unschedulable system here is an issue. There can be three obvious choices: forward order, backward order, and check the deadlines by each task in turn.

The first two choices, forward order and backward order, need all absolute deadlines to be sorted, using the fastest approach, sorting m absolute deadlines has the complexity $O(m \log_2 m)$ in average, and $O(m^2)$ in the worst-case, when m is large, the time needed to spent on the pretreatment of sorting could be far more than the schedulability test itself. If we compare QPA with the previous results by these two orders, it is unfair for the proposed analysis, since the QPA algorithm does not need such pretreatment.

Despite these factors, we would also like to observe what order to be used by previous results can find a failure point with the least number of checked deadlines. We investigate this first.

(H) Frequency of the first failure absolute deadline

These experiments explore the frequency distribution of how many absolute deadlines have been checked when the program first finds a failure point according to different checking orders.

We denote N_{ch} to be the number of deadlines that have been checked when the we first find a missed deadline for a task set, and denote N_{to} to be the total number of deadlines in $L = \min\{L_a^*, L_b\}$. The value of N_{ch} / N_{to} inverses with the number of checked deadlines.

The following 3 experiments are based on 1000 unschedulable task sets with the utilization 0.9, and the maximum value of $T_{\max} / T_{\min}$ is 1000. On the x-axis, 0.1 means

$0 < N_{ch} / N_{to} \le 0.1$, and so on.

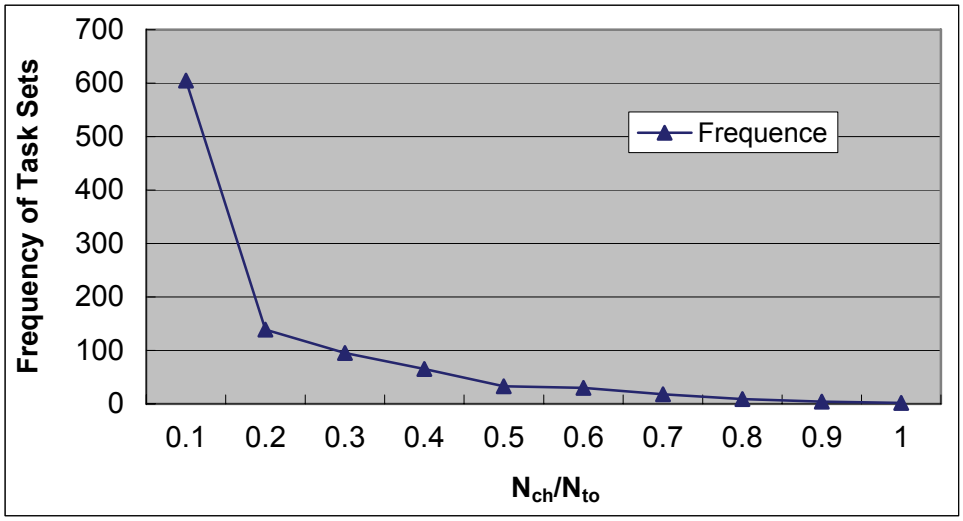

Figure 17. Frequency of the forward order

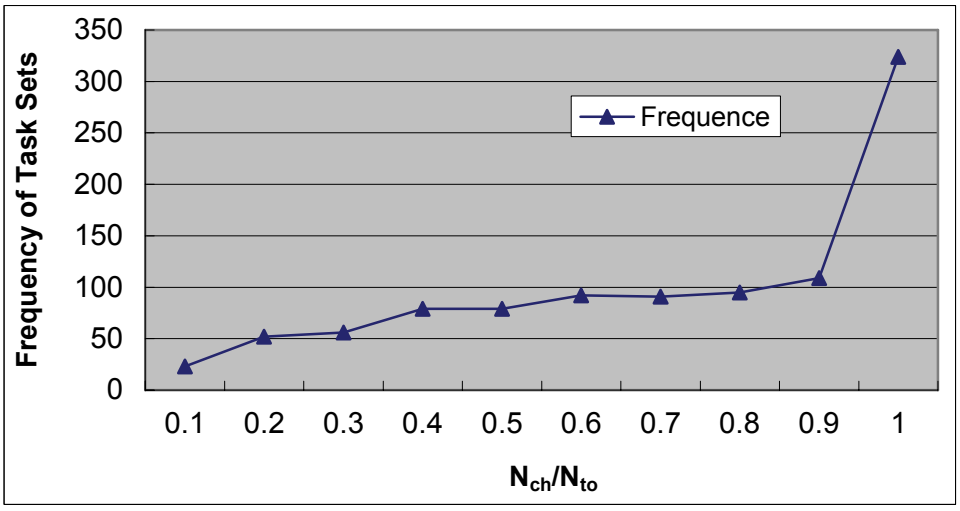

Figure 18. Frequency of the backward order

This experiment which explores the frequency of checking by each task in turn, the order of tasks is checked by $\tau_1, \tau_2, \tau_3, \ldots$, from the default generation policy, the less number task have the more chance to get a shorter relative deadline (see example B, Section 6), so the order can be regarded as smaller relative deadline task checked first.

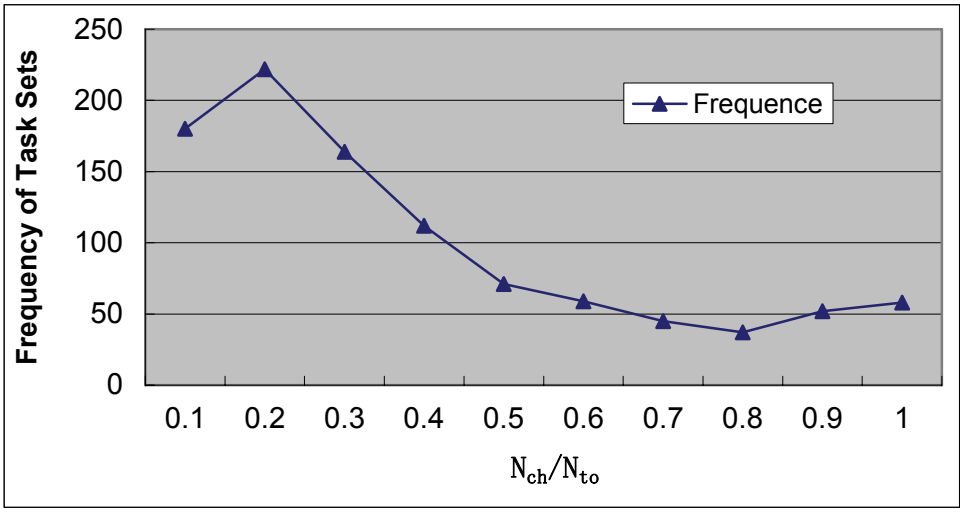

Figure 19. Frequency of checking by each task (order of task)

From the above experiments, we can see that verify forward from $t = 0$ for the sorted absolute deadlines can find an overflow with the least number of checked deadlines.

Although only comparing how many deadlines have been checked is unfair for the new results, we would still like to compare QPA with the previous results by forward order from $t = 0$. In the following experimental comparisons, we ignore all the additional calculations required by previous results such as sorting the deadlines.

When all task sets are schedulable, by the previous results, all the absolute deadlines in an upper bound have to be verified, so we only need to count here are how many deadlines in each upper bound in an experiment. But when the task sets are unschedulable, we need to sort deadlines, and we need to check every deadline until we find a failure one, due to the large amount of calculation required by the previous results, in this section, we could not experiment on so large a range of the task sets as the experiments in the previous section.

Except experiment (N), each point on the following diagrams are the average of 2,000 randomly generated task sets which are unschedulable. The "old method" on the graph means the number of absolute deadlines which have been checked by the previous analysis by the forward order.

(I) Impact of the task periods range

In this experiment, the utilization of each task set is 0.9, and task number of each task set is 30.

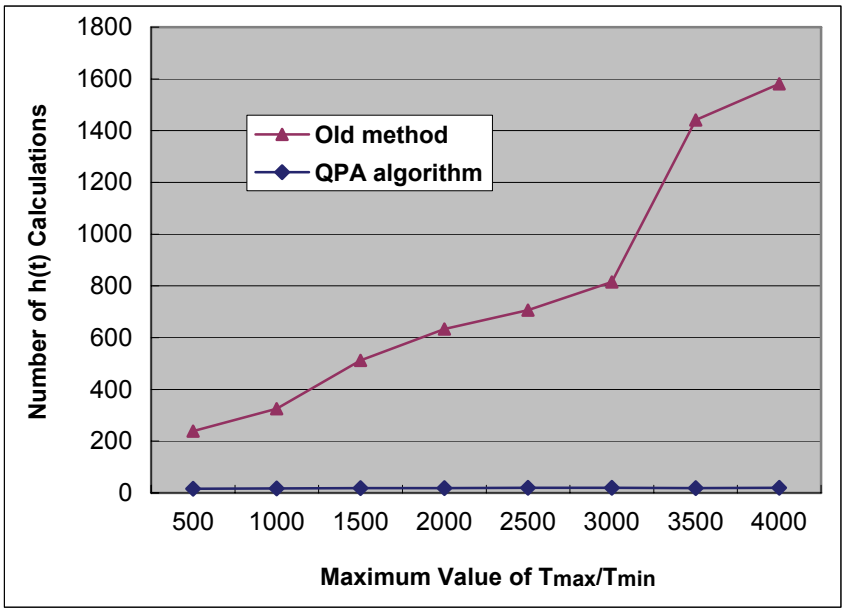

Figure 20. Impact of the task periods range

The default value of the maximum value of $T_{\max} / T_{\min}$ for all the experiments in this section is set to be 1000. From this experiment, we can see that if we let the maximum value of $T_{\max} / T_{\min}$ be larger than 1000, then the experimental results for the previous analysis in all experiments will be increased significantly.

(J) Impact of the tasks number

In this experiment, we set the utilization of each task set to be 0.9, and for each task set, the maximum value of $T_{\max} / T_{\min}$ is 1000.

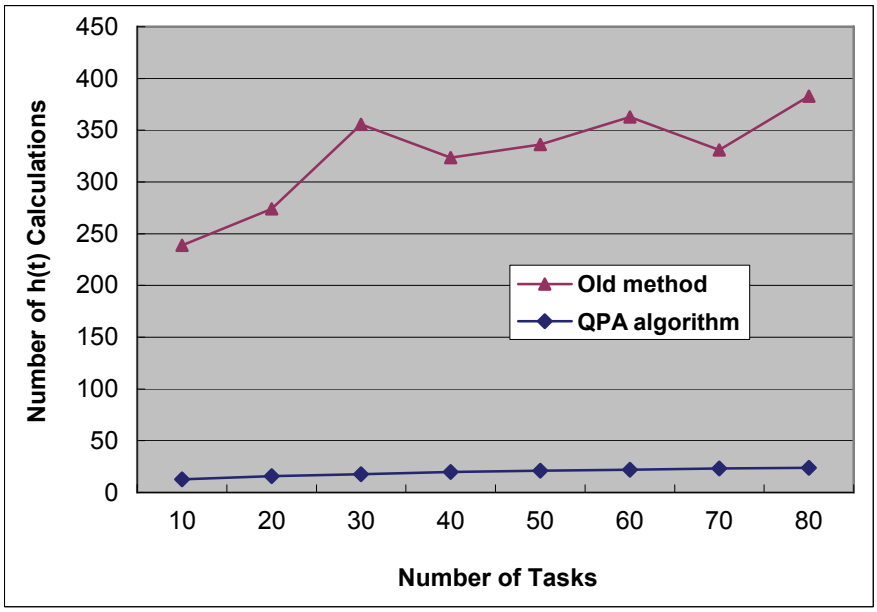

Figure 21. Impact of the number of tasks

(K) Impact of the task set's utilization

In this experiment, we let tasks number of each task set be 30, and the maximum value of T_{max} / T_{min} be 1000.

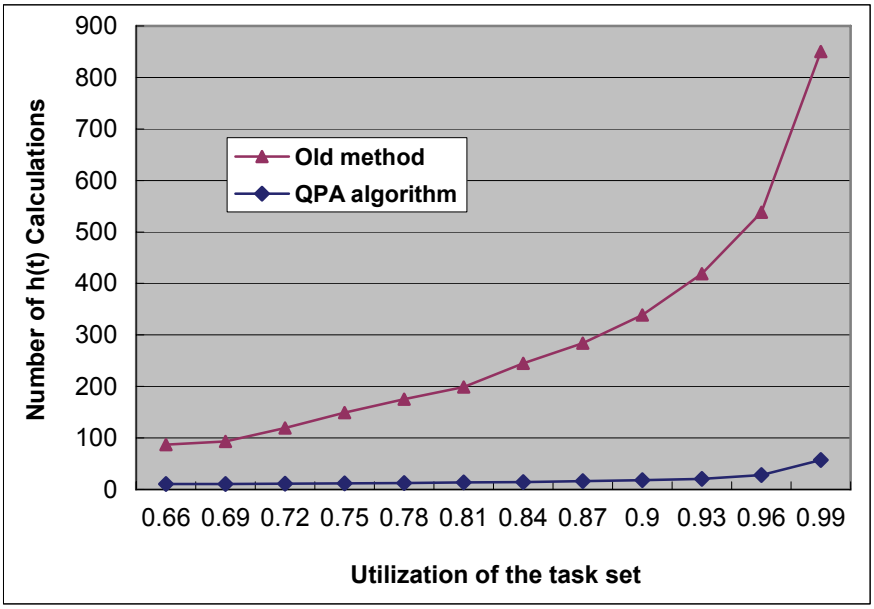

Figure 22. Impact of the task set's utilization

(L) Impact of the fixed value of D_i / T_i

In this experiment, we change the default generation policy of D_i, let D_i / T_i be a fixed value k, hence $D_i = k \times T_i$. We set the tasks number of each task set to be 30, each task set's utilization to be 0.9, and the maximum value of T_{max} / T_{min} to be 1000.

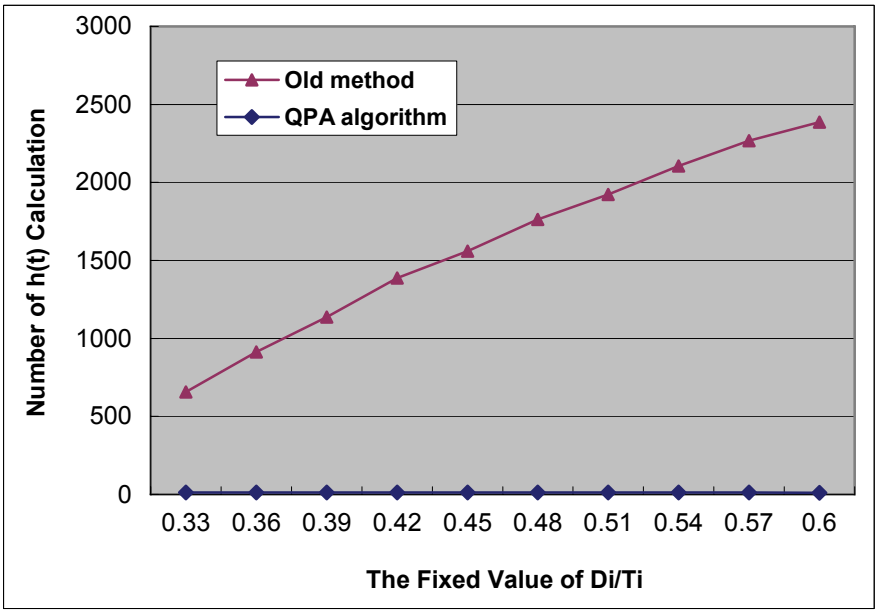

Figure 23. Impact of the fixed value of Di/Ti

Let the number of the schedulable tasksets be N_{sche}, then the percentage of the schedulable task sets under each value D_i / T_i is:

$$percentage = \frac{N_{sche}}{N_{sche} + 2000} \times 100\% .$$

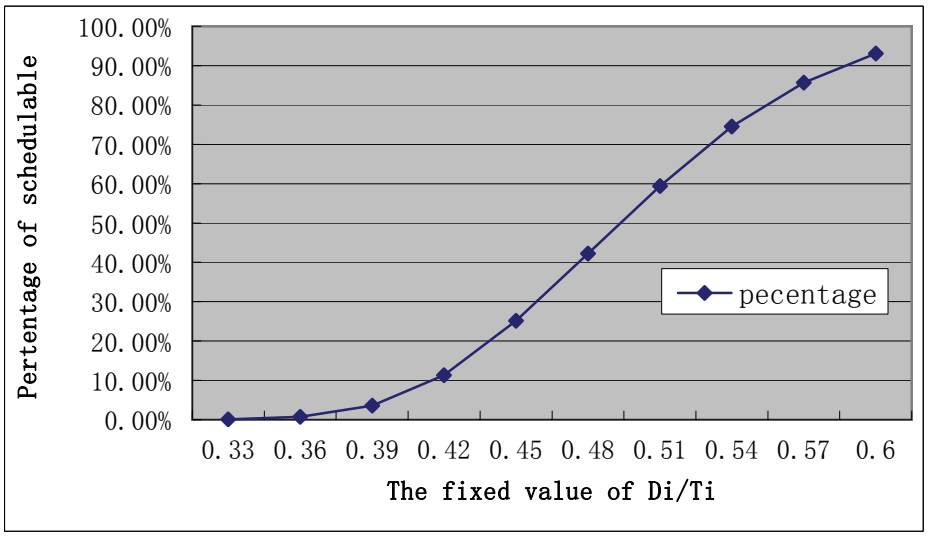

Figure 24. Percentage of the schedulable task sets

From Figure 24, when each $D_i / T_i \le 0.36$, nearly all task sets are unschedulable, so we should not use such task sets, and the experimental comparisons on such task sets is meaningless. Only $D_i / T_i > 0.36$ when there are some task sets are schedulable, the comparisons then become meaningful. In this situation, the required number of $h(t)$ calculations by the old result is close to 1000.

(M) Control the distance from each D_i to C_i

In this experiment, we control the distance between each D_i and C_i, let each D_i be generated from $a = \min\{D_i / C_i\}$ to b, where b remains the default value of the generation policy.

For each task set, the utilization is 0.9, the number of tasks is 30, and the maximum value of $T_{\max} / T_{\min}$ is 1000.

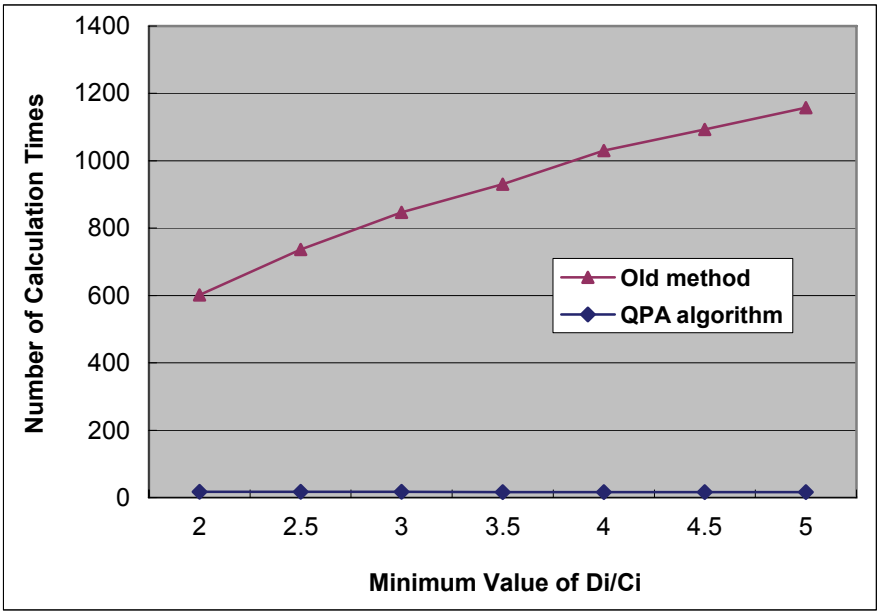

Figure 25. Impact of the minimum value of Di/Ci

Percentage of the schedulable task sets:

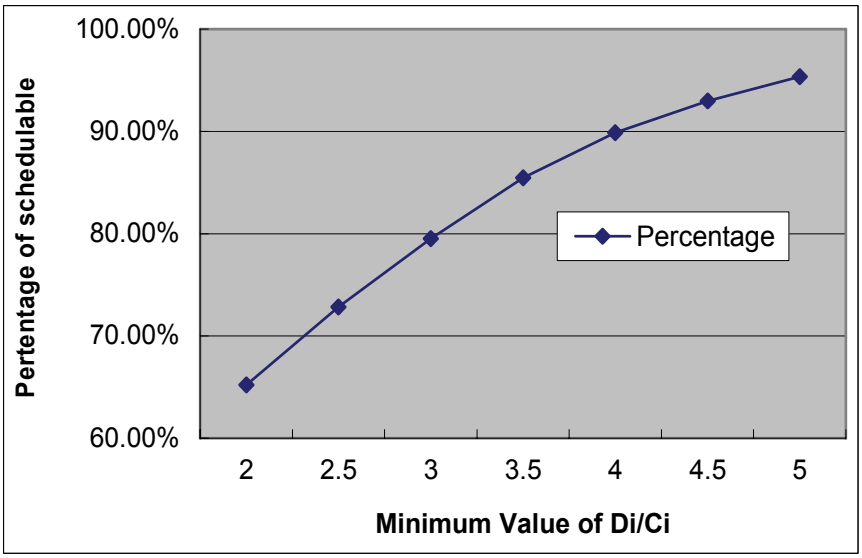

Figure 26. Percentage of the schedulable task sets

Compare experiments (L)(M) with experiments (I)(J)(K), we can find that when the distance between each D_i and C_i is no less than a certain value (i.e. $2 \times C_i$), the calculation times by the previous results is increased significantly, and we can also reach the conclusion that under the same situation (i.e. $n = 30$, $u = 0.9$, task periods range=1000), the number of calculation times in experiments (I)(J)(K) is less than the results of (L)(M), that's because in (I)(J)(K) some D_i s are too close to C_i s, an overflow can often be found at the beginning of the deadline checking by forward order. So the

default generation policy and the experimental results in (I)(J)(K) are optimistic for the old analysis.

(N) Frequency distribution of the task sets

This experiment based on 60,000 randomly generated task sets which are unschedulable, for each task set, the utilization is 0.9, the task number is 30, and the maximum value of $T_{\max} / T_{\min}$ is 1000.

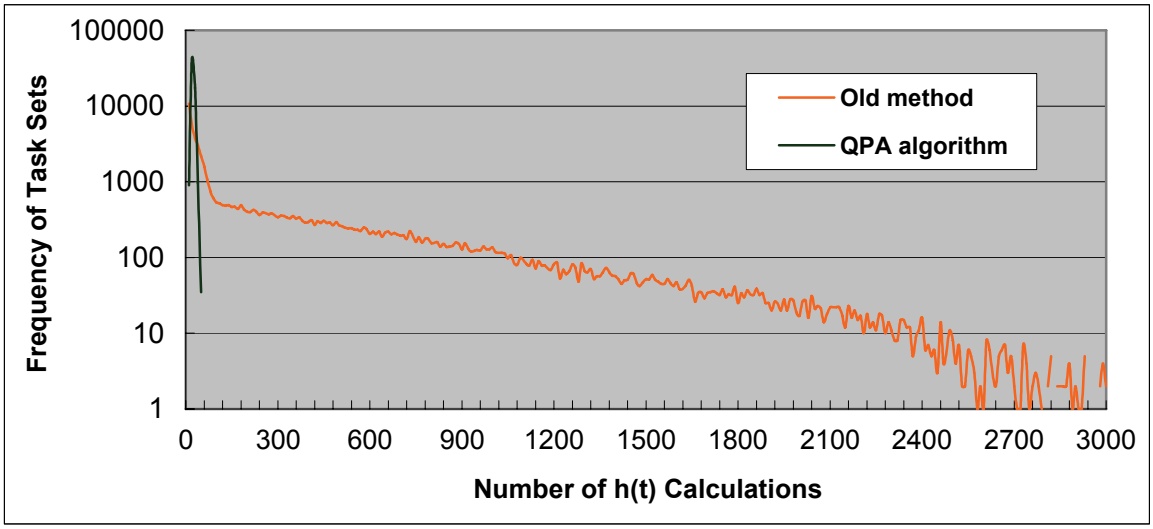

Figure 27. Frequency distribution based on 60,000 unschedulable task sets

The following graph is the frequency distribution of the calculation required by QPA according to the above experiment.

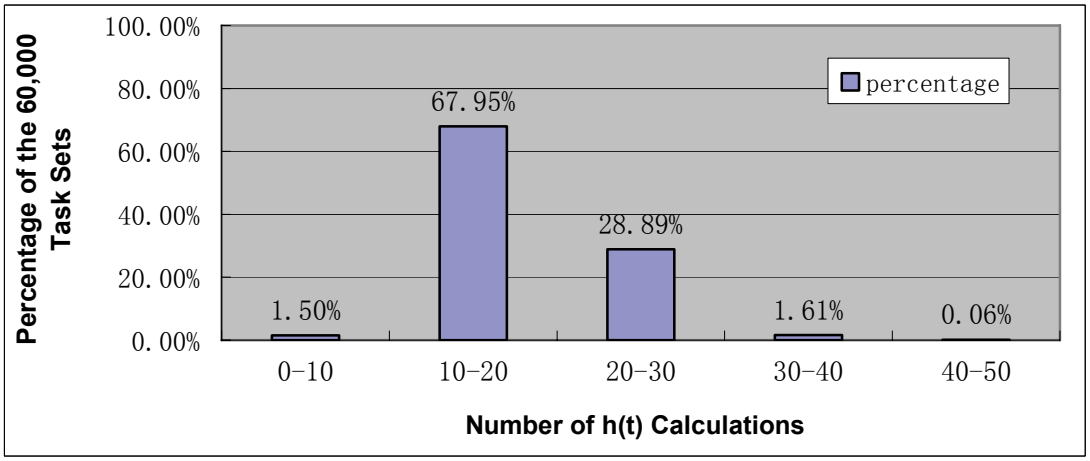

Figure 28. Frequency distribution of QPA

7.3 Conclusion of the experiments

From the experiments describes in this section, we can see that a lot of factors can

significantly affect the experimental results of the old methods; in some circumstances, they have exponential growth. The experimental results of QPA are stable for all kinds of task sets, and QPA reduces the required calculation in logarithm scales in all situations.

In experiment (G) based on 80,000 and experiment (N) based on 60,000 randomly generated task sets, by the QPA algorithm, more than 96% of the task sets complete each schedulability test in less than 30 calculation times of $h(t)$, and all the task sets complete each schedulability test in less than 60 calculation of $h(t)$. The function $h(t)$ has the complex $O(n)$, equal to calculating a task set's utilization. This means the vast majority of the task sets in the experiments only require the calculation which is equivalent to less than 30 times the utilization based test.

We also observed that the new upper bound L_a^* dominants the old results when each D_i is no larger than $2T_i$. The calculation of L_b has an iterative process which may need more iteration times than the QPA algorithm, since L_a^* is simpler to calculate than L_b, we would suggest that only L_a^* is used in the QPA when each D_i is not too larger than $2T_i$.

8. Schedulability Analysis with Release Jitter

In the previous literature, Spuri [14] mentioned if we remove the assumption of null release jitter, then the worst-case arrival pattern for testing the tasks schedulability by processor demand criterion is obtained by releasing the first instance of each task at time $t = 0$, and all other instances are released at $t = \max\{kT_i - J_i, 0\}$. However this conclusion was not formally proved. Under this worst-case arrival pattern, the processor demand $h_J(t)$ in a given interval is calculated by [14]:

$$h_J(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i = \sum_{D_i \le t + J_i} \left\lfloor \frac{t + T_i + J_i - D_i}{T_i} \right\rfloor C_i, \tag{9}$$

and the length of the synchronous busy period is calculated by:

$$w^{m+1} = \sum_{i=1}^{n} \left\lceil \frac{w^m + J_i}{T_i} \right\rceil C_i, \tag{10}$$

Here we give the above assumption a formal proof, and provide a complete schedulability test which is necessary and sufficient for EDF scheduling which considers release jitter and incorporates the new bound discussed in Section 4.

Theorem 8 A general task set is schedulable by EDF if and only if all the following conditions are true:

1) $U \le 1$.

2) All tasks are released simultaneously at time $t = 0$ after having experienced their

maximum release jitter, and then the subsequent instances of each task are released at their maximum rate.

3) $\forall t \in P \quad h(t) \le t$,

where $h_J(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i$,

and P is the set of absolute deadlines in $\min(L_a^J, L_b^J)$, that is:

$$P = \{d_i \mid d_i = kT_i + D_i - J_i \wedge d_i < \min(L_a^J, L_b^J), k \in N\}, \quad \text{where}$$

$$L_a^J = \max\{(D_1 - T_1 - J_1), ..., (D_n - T_n - J_n), \frac{\sum_{i=1}^{n} (T_i + J_i - D_i) U_i}{1 - U}\},$$

and L_b^J is the synchronous busy period with the arrival pattern of condition 2 (i.e. solution of equation (10)).

Proof. We start from condition 2 which gives the worst-case arrival pattern for task schedulability and the longest synchronous busy period. To prove this, at first, we remove the release jitter assumption. Let t_1 be an overflow time in any tasks arrival pattern, and t be the last time before t_1 such that there are no pending jobs with absolute deadline $D_i \le t_1 - t$ before t_1. If we move "left" all tasks' first instances which arrive after t, let all tasks be released simultaneously at t, the task load in $[t, t_1]$ could only be increased, so there is still an overflow at or before t_1, let t_2 denote the new overflow time after the shift, we have $t_2 \le t_1$.

Then we add the release jitter assumption. If we let the release time of the first instance (arrives at time t) of each task τ_i be fixed, and move "left" all other instances of task τ_i in the period $[t, t_2]$, making them closer to the first one, obviously the more we move and make the second and the following instances closer to the first one, the task load in the period $[t, t_2]$ could become larger. The maximum distance of each task τ_i's other instances can be moved forward to the first one is J_i, that is the situation when all tasks arrive simultaneously at time t after having experienced their maximum release jitter, and then the subsequent instances released at their maximum rate. So the arrival pattern of condition 2 is the worst-case for testing the tasks schedulability, and the overflow still occurs at or before t_2, let t_3 denote the new overflow time when the tasks arrival pattern has been changed to condition 2, then we have $t_3 \le t_2$.

From Lemma 2, there is no processor idle time during $[t, t_1]$; from Lemma 1, L_b gives

the longest busy period without considering release jitter, so $t_1 - t < L_b$. We can apply the same argument of the previous paragraph to show that after changing the tasks arrival pattern to condition 2, the length of any busy period could only be increased, hence the arrival pattern of condition 2 also gives the maximum length of any busy period. Let L_b^J denote the synchronous busy period with the arrival pattern of condition 2, we have $L_b \le L_b^J$. As $t_3 \le t_2 \le t_1$, we have $t_3 - t \le t_1 - t < L_b \le L_b^J$. Let $t = 0$, $t_3 \le t_1 < L_b^J$, so the overflow must be found in the period $(0, L_b^J)$.

When $t \ge \max_{1 \le i \le n}\{D_i - T_i - J_i\} \quad \Leftrightarrow \quad t \ge D_i - T_i - J_i \quad \Leftrightarrow \quad t + J_i - D_i \ge -T_i \quad \Leftrightarrow$

$\left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \ge -1 \quad \Leftrightarrow \quad 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \ge 0$,

then we have: $h(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i = \sum_{i=1}^{n} (1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor) C_i$

$$= t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\le t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

If $U \le 1$ and the task set is not schedulable, $t < h(t)$

$$\Leftrightarrow \qquad t < t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow \qquad t(1 - \sum_{i=1}^{n} \frac{C_i}{T_i}) < \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow \qquad t < \frac{\sum_{i=1}^{n} (T_i + J_i - D_i) U_i}{1 - U} \qquad \square$$

Integrate with the new algorithm

In order to corporate release jitter into the new algorithm in Section 5, we let:

$$L = \begin{cases} \min(L_a^J, L_b^J) & u < 1 \\ L_b^J & u = 1 \end{cases}, \qquad h(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i,$$

and $d_{\min} = \min_{1 \le i \le n}\{D_i - J_i\}$, then we can use Theorem 7 to test schedulability.

On the occasion when we need to find $\max\{d_i \mid d_i \le t\}$, for a single task τ_k, the last

arrived job before time t is released at:

$$(\left\lceil \frac{t+J_k}{T_k} \right\rceil -1)T_k ,$$

with the absolute deadline:

$$d_k = (\left\lceil \frac{t+J_k}{T_i} \right\rceil -1)T_k + D_k ,$$

so $\max\{d_i \mid d_i \le t\}$ can be calculated by:

---

for $(k=1; k \le n; k++)$

$\{ d_k = (\lceil (t+J_k)/T_k \rceil -1)T_k + D_k ;$

while $(d_k > t)$ $d_k = d_k - T_k ;$

if $(d_i > d_{\max}^t)$ $d_{\max}^t = d_k ;$

$\}$

After the recurrence, $d_{\max}^t = \max\{d_i \mid d_i \le t\}$.

9. Resource Sharing and Release Jitter

In typical realistic cases, the tasks in the system are not independent and they need to access shared non-preemptable resources. If a high-priority process is suspended waiting for a lower-priority task to complete its use of a non-preemptable resource, then priority inversion [8] occurs, it is said that the high-priority process is blocked by the lower priority process.

There are a number of protocols that exist for limiting this priority inversion, the most important and widely used is the Stack Resource Policy (SRP) [1], which extends the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP) [13]. SRP is very easily integrated into the EDF scheduling framework.

In this section, under the framework of EDF+SRP, we show that the blocking factor and release jitter can be considered at the same time into an exact schedulability analysis, and the new results can be integrated with both release jitter and blocking.

9.1 Conditions for the Schedulability

Let $\pi(\tau_j)$ denote the preemption level of a task τ_j, $\pi(\tau_j) > \pi(\tau_k) \Leftrightarrow D_j < D_k$. Each shared resource R_i is assigned a ceiling $\Pi(R_i)$ which is set equal to the maximum

preemption level of any task that may access it. Let $\overline{\pi}$ denote the highest ceiling of all the resources which are held by some tasks at any given time, that is: $\overline{\pi} = \max\{\Pi(R_i) \mid R_i \text{ is held}\}$. Baker [1] showed the Stack Resource Policy has the following properties:

Theorem 9 ([1]) If no job r_j is permitted to start execution until $\pi(\tau_j) > \overline{\pi}$, then:

1) no job can be blocked after it starts;
2) there can be no transitive blocking or deadlock;
3) no job can be blocked for longer than the execution time of one outermost critical section of a lower priority job;
4) if the oldest highest-priority job is blocked, it will become unblocked no later than the first instant that the currently executing job is not holding any non-preemptable resources.

Let $B(t)$ be a function presents the maximum time for job r_k with relative deadline $D_k \le t$ may be blocked by job r_α with relative deadline $D_\alpha > t$ in any given time interval $[0,t]$.

Spuri [14] showed that a sufficient condition for schedulability of a task set is that for any absolute deadline d_i in a synchronous busy period:

$$h(d_i) + B(d_i) \le d_i . \tag{11}$$

Although the definition of $B(t)$ given by Spuri [14] implies that release jitter is considered, in the proof of the schedulability condition they did not take account of release jitter, only the blocking factor is incorporated.

The definition of $B(t)$ given by Baruah [5] is more intuitive: let $C_{\alpha,k}$ denote the maximum length of time for which task τ_α needs to hold some resource that may also be needed by task τ_k, then $B(t)$ can be defined and calculated by:

$$B(t) = \max\{C_{\alpha,k} \mid D_\alpha > t\ , D_k \le t\} . \tag{12}$$

In the definitions of the remaining part of this section, to incorporate release jitter, we suppose $\alpha \ne k$.

Here we define $B_J(t)$ as:

$$B_J(t) = \max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t\ ,\ \ D_k - J_k \le t\} , \tag{13}$$

note $B_J(t)$ can also be defined as:

$$B_J(t) = \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha\ , D_k \le t + J_k\} . \tag{14}$$

$B_J(t)$ is defined under the assumption when all tasks experienced their maximum release jitter.

If each task τ_i does not experience its maximum release jitter, but only experience a

jitter equals to J_i', where J_i' is an random number in $[0, J_i]$, then the blocking time should be defined and calculated as:

$$B_{J'}(t) = \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha',\ D_k \le t + J_k'\}, \tag{15}$$

where $0 \le J_\alpha' \le J_\alpha$, and $0 \le J_k' \le J_k$.

So the definition of $B(t)$ and the definition of $B_J(t)$ are special cases of $B_{J'}(t)$.

Under the assumption that each task τ_i only experience a jitter equals to J_i', where $J_i' \in [0, J_i]$, we can apply the same argument in Theorem 8's proof to show that the worst-case arrival pattern for schedulability occurs when each task τ_i is released at time 0 after having experienced a jitter J_i', then at its maximum rate. In this task arrival pattern, the processor demand in a given time interval $[0, t]$ becomes:

$$h_{J'}(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i' - D_i}{T_i} \right\rfloor\} C_i, \tag{16}$$

and the processor demand plus the blocking time in $[0, t]$ is calculated by $h_{J'}(t) + B_{J'}(t)$.

The maximum blocking time depends on how many jobs can be blocked and how many jobs can block other jobs. Comparing the definitions of $B_J(t)$ and $B_{J'}(t)$, we can see that in the definition of $B_J(t)$, although the number of jobs which could be blocked is increased (from $D_k \le t + J_k$), the number of jobs which could block other jobs is decreased (from $D_\alpha > t + J_\alpha$). So at a given t, the value of $B_J(t)$ can be less than $B_{J'}(t)$. In other words, when all tasks experience their maximum jitter, the maximum blocking time could be decreased. In order to show condition 2 of Theorem 8 is still the worst-case arrival pattern for schedulability after considering the blocking time, we need to prove at any given time t, $h_{J'}(t) + B_{J'}(t) \le h_J(t) + B_J(t)$.

Lemma 6 $\forall t > 0$, $h_{J'}(t) + B_{J'}(t) \le h_J(t) + B_J(t)$

Proof. If $B_{J'}(t) \le B_J(t)$, since $h_{J'}(t) \le h_J(t)$, we have $h_{J'}(t) + B_{J'}(t) \le h_J(t) + B_J(t)$. Then we only need to prove the situation when $B_{J'}(t) > B_J(t)$.

From the definition of $B_J(t)$ and $B_{J'}(t)$, $B_J(t) < B_{J'}(t) \Rightarrow$

$$\max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha, D_k \le t + J_k\} < \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t + J_k'\}. \tag{17}$$

At any given t, we always have:

$$\max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t\} \le \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t + J_k\}. \tag{18}$$

From inequation (17)(18):

$$\max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha, D_k \le t + J_k\} < \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t + J_k\}. \tag{19}$$

From inequation (19), there exists a task τ_γ which satisfies all the following conditions:

1) denote D_γ to be its relative deadline, J_γ to be its maximum release jitter, and

J_γ' to be a jitter between 0 and J_γ, then $t + J_\gamma' < D_\gamma \le t + J_\gamma$;

2) task τ_γ needs to hold some shared non-preemptive resources that are also needed by a task τ_k with $D_k \le t + J_k$, $\gamma \ne k$, denote $C_{\gamma,k}$ to be the maximum length of such resource;

3) $C_{\gamma,k}$ is no shorter than any other shared resource in

$$\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t + J_k\}.$$

From condition 1:

$$D_\gamma \le t + J_\gamma \;\Rightarrow\; t + J_\gamma - D_\gamma \ge 0 \;\Rightarrow\; (1 + \left\lfloor \frac{t + J_\gamma - D_\gamma}{T_\gamma} \right\rfloor) C_\gamma \ge C_\gamma.$$

$$t + J_\gamma' < D_\gamma \;\Rightarrow\; t + J_\gamma' - D_\gamma \;\Rightarrow\; (1 + \left\lfloor \frac{t + J_\gamma' - D_\gamma}{T_\gamma} \right\rfloor) C_\gamma = 0.$$

Hence:

$$h_J(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i$$

$$\ge \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i' - D_i}{T_i} \right\rfloor\} C_i + C_\gamma$$

$$= h_{J'}(t) + C_\gamma.$$

From condition 3: $C_{\gamma,k} = \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t + J_k\}$

$$\ge \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha', D_k \le t + J_k'\}$$

$$= B_{J'}(t).$$

Since $C_\gamma \ge C_{\gamma,k} \ge B_{J'}(t)$, we have:

$$h_J(t) + B_J(t) \ge h_J(t) \ge h_{J'}(t) + C_\gamma \ge h_{J'}(t) + B_{J'}(t). \qquad \square$$

Theorem 10 A general task set is schedulable by EDF+SRP if the following conditions are true:

1) $U \le 1$.

2) All tasks are released simultaneously at time $t = 0$ after having experienced their maximum release jitter, and then the subsequent instances of each task are released at their maximum rate.

3) $\forall t \in P \quad h_J(t) + B_J(t) \le t$,

where $h_J(t) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i$, $B_J(t) = \max\{C_{\alpha,k} \mid D_\alpha > t + J_\alpha, D_k \le t + J_k\}$,

and P is the set of absolute deadlines in $\min(L_a^B, L_b^J)$, that is:

$$P = \{d_i \mid d_i = kT_i + D_i - J_i \wedge d_i < \min(L_a^B, L_b^J), k \in N\}, \quad \text{where}$$

$$L_a^B = \max\{(D_1 - T_1 - J_1), ..., (D_n - T_n - J_n), \frac{\max_{d_i < D_{\max}}\{B_J(d_i)\} + \sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1 - U}\}, \qquad (20)$$

where $D_{\max} = \max_{1 \le i \le n}\{D_i - J_i\}$, and L_b^J is the synchronous busy period with the arrival pattern of condition 2.

Proof. We start from condition 3, and show that the task set is schedulable if at any deadline d_i, $h_J(d_i) + B_J(d_i) \le d_i$. At first we assume there is no release jitter, let t be a time when there is a job miss its deadline in any tasks arrival, and $t' = 0$ be the last time before t such that there are no pending jobs with deadlines less than or equal to t. At time $t' = 0$, if there is an arrived job r_α with relative deadline $D_\alpha > t$ holding some shared resources which are also needed by the later arrived jobs r_k with $D_k \le t$, then this low priority job r_α may block a higher priority (earlier deadline) job r_k. Let t_u be the first time when job r_α complete its access for a shared resources and does not hold any shared resource, from Theorem 9, the oldest highest-priority blocked job will become unblocked immediately at time t_u. Then there are always pending jobs with deadlines less than or equal to t in the period $[t_u, t]$, no earlier deadline jobs can be blocked again after t_u, and r_α is the only instance which could be executing in $[0, t]$ with absolute deadline larger than t. The maximum time in which job r_α could be executing in the time interval $[0, t]$ is $C_{\alpha,k}$, so the maximum blocking time in $[0, t]$ is $B(t) = \max\{C_{\alpha,k} \mid D_\alpha > t, D_k \le t\}$.

Then we change the tasks arrival pattern. Let each task with $D_k \le t + J_k$ be released simultaneously at time $t = 0$ after having experienced its maximum release jitter then at its maximum rate, and let each task with $D_\alpha > t + J_\alpha$ be released at time $-\varepsilon$ after experiencing the release jitter equal to $J_\alpha - \varepsilon$ then at its maximum rate, where ε is a infinite small number. For each task with $D_\alpha > t + J_\alpha$, each absolute deadline of its job is $kT_\alpha + D_\alpha + (-\varepsilon) - (J_\alpha - \varepsilon) = kT_\alpha + D_\alpha - J_\alpha, k \in N$. Then we get an arrival pattern of condition 2, from Theorem 8's proof, this arrival pattern is the worst-case for tasks schedulability without considering the blocking factor. Since each task with $D_\alpha > t + J_\alpha$ is

released ε time units before the release time of each task with $D_k \le t + J_k$, any task with $D_k \le t + J_k$ can be blocked by any task with $D_\alpha > t + J_\alpha$ if they need to hold the same non-preemptable resource, then the maximum blocking time in $[0,t]$ becomes: $B_J(t) = \max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t, D_k - J_k \le t\}$. After adding the blocking time, although the blocking time could be decreased under this arrival pattern, from Lemma 6, the total task load ($h_J(t) + B_J(t)$) could only be increased. Therefore there is still an overflow at or before time t, and condition 2 is still the worst-case arrival pattern for the tasks schedulability under EDF+SRP.

Since both of the value of $h_J(t)$ and $B_J(t)$ could only be changed at an absolute deadline $d_i = kT_i + D_i - J_i, k \in N$, if $\forall d_i$, $h_J(d_i) + B_J(d_i) \le d_i$, then there is no overflow at any time.

The following part of the proof is to reduce the period we need to look for. As the blocking only changes the execution order of the jobs in a busy period, any overflow still occurs in a busy period, and the length of the longest busy period of the tasks is not changed, from Theorem 8's proof, condition 2 gives the maximum length of any busy period, so we can still use L_b^J as an upper bound for the schedulability test.

When $t \ge \max_{1 \le i \le n}\{D_i - T_i - J_i\} \Leftrightarrow t \ge D_i - T_i - J_i \Leftrightarrow t + J_i - D_i \ge -T_i \Leftrightarrow$

$$\left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \ge -1 \Leftrightarrow 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor \ge 0,$$

then we have: $h_J(t) + B_J(t) = B_J(t) + \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor\} C_i$

$$= B_J(t) + \sum_{i=1}^{n} (1 + \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor) C_i$$

$$= B_J(t) + t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\le \max_{d_i < D_{\max}}\{B_J(d_i)\} + t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

If $U \le 1$ and the task set is not schedulable, $t < h_J(t) + B_J(t)$

$$\Leftrightarrow \quad t < \max_{d_i < D_{\max}}\{B_J(d_i)\} + t\sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow \quad t(1-\sum_{i=1}^{n}\frac{C_i}{T_i}) < \max_{d_i<D_{\max}}\{B_J(d_i)\} + \sum_{i=1}^{n}\frac{C_i}{T_i}(T_i + J_i - D_i)$$

$$\Leftrightarrow \quad t < \frac{\max_{d_i<D_{\max}}\{B_J(d_i)\} + \sum_{i=1}^{n}(T_i + J_i - D_i)U_i}{1-U} \qquad \square$$

Like all the previous results on the task schedulability analysis which incorporates SRP, since the definition and calculation of $B(t)$ may be pessimistic in some cases, the above analysis is only sufficient for schedulability under EDF+SRP.

In equation (20) of Theorem 10, $\max_{d_i<D_{\max}}\{B(d_i)\}$ can be obtained by calculating the value of $B(t)$ at every $d_i \in (0, D_{\max})$. As $\max_{d_i<D_{\max}}\{B_J(d_i)\} \le \max_{1\le i\le n}\{C_i\}$, for convenience, we can also use $\max_{1\le i\le n}\{C_i\}$ as $\max_{d_i<D_{\max}}\{B_J(d_i)\}$.

9.2 Integrate with the New Algorithm

In this section, we integrate blocking and release jitter into the algorithm of Section 5, so that tasks with release jitter can share non-preemptable resources in our system model.

To give the new theorem which incorporates blocking and release jitter, we need to show that $h_J(t) + B_J(t)$ is non-decreasing with t.

Lemma 7 $\forall t > 0$, $h_J(t) + B_J(t)$ is a non-decreasing function of t.

Proof. $\forall t_1, t_2, 0 < t_1 < t_2$, when $B_J(t_1) \le B_J(t_2)$, $h_J(t_1) + B_J(t_1) \le h_J(t_2) + B_J(t_2)$, so we only need to consider the situation when $B_J(t_1) > B_J(t_2)$.

From the definition of $B_J(t)$, $B_J(t_1) > B_J(t_2) \Rightarrow$

$$\max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_1, D_k - J_k < t_1\} > \max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_2, D_k - J_k < t_2\}.$$

For any $t_1 < t_2$, we have:

$$\max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_2, D_k - J_k < t_2\} \ge \max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_2, D_k - J_k < t_1\}.$$

Therefore:

$$\max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_1, D_k - J_k < t_1\} > \max\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_2, D_k - J_k < t_1\} \qquad (21)$$

From inequation (21), there exists a task τ_ξ which satisfies all of the following condition:

1) $t_1 < D_\xi - J_\xi \le t_2$;
2) task τ_ξ needs to hold some non-preemptable resource that is also needed by a task τ_k with $D_k - J_k \le t_1$, denote $C_{\xi,k}$ to be the maximum length of such a

resource access;

3) $C_{\xi,k}$ is no shorter than any other shared resources in

$$\{C_{\alpha,k} \mid D_\alpha - J_\alpha > t_1, D_k - J_k \le t_1\},$$

this means $C_{\xi,k} = B_J(t_1)$.

Let the execution time of task τ_ξ be C_ξ, we have $B_J(t_1) = C_{\xi,k} \le C_\xi$.

As $t_1 < D_\xi - J_\xi$, $t_1 + J_\xi - D_\xi < 0 \Rightarrow (1 + \left\lfloor \frac{t_1 + J_\xi - D_\xi}{T_\xi} \right\rfloor) C_\xi = 0$ (22)

As $D_\xi - J_\xi \le t_2$, $t_2 + J_\xi - D_\xi \ge 0 \Rightarrow (1 + \left\lfloor \frac{t_2 + J_\xi - D_\xi}{T_\xi} \right\rfloor) C_\xi \ge C_\xi$ (23)

From (22)(23): $h(t_2) = \sum_{i=1}^{n} \max\{0, 1 + \left\lfloor \frac{t_2 + J_i - D_i}{T_i} \right\rfloor\} C_i \ge h(t_1) + C_\xi$

Therefore we have:

$$h_J(t_1) + B_J(t_1) = h_J(t_1) + C_{\xi,k} \le h_J(t_1) + C_\xi \le h_J(t_2) \le h_J(t_2) + B_J(t_2). \quad \square$$

After getting Lemma 7, Lemma 8, Lemma 9, and Theorem 11 can be proved, so both blocking and release jitter can be integrated together into the algorithm of Section 5.

In the remaining part of this section, we define L as:

$$L = \begin{cases} \min(L_a^B, L_b^J) & U < 1 \\ L_b^J & U = 1 \end{cases},$$

and when a system is not judged to be schedulable, denote

$$d^\Delta = \max\{d_i \mid d_i \in (0, L) \wedge d_i < h_J(d_i) + B_J(d_i)\}.$$

Lemma 8 For a system that is not judged to be schedulable, let $d_m = \max\{d_i \mid 0 < d_i < L\}$. If $h_J(d_m) + B_J(d_m) \le d_m$, then $d^\Delta < h_J(d^\Delta) + B_J(d^\Delta) \le d'$, where $d' = \min\{d_i \mid d_i > d^\Delta\}$.

Proof. Since $d_m = \max\{d_i \mid 0 < d_i < L\}$, and $h_J(d_m) + B_J(d_m) \le d_m$, we have $d^\Delta < d_m$, and $d' \le d_m < L$. Suppose $h_J(d^\Delta) + B_J(d^\Delta) > d'$, since $d' > d^\Delta$, and from Lemma 7, $h_J(t) + B_J(t)$ is non-decreasing with t, so we have:

$$h_J(d') + B_J(d') \ge h_J(d^\Delta) + B_J(d^\Delta) > d'.$$

This contradicts the condition that $d^\Delta = \max\{d_i \mid d_i \in (0, L) \wedge d_i < h_J(d_i) + B_J(d_i)\}$, therefore $d^\Delta < h_J(d^\Delta) + B_J(d^\Delta) \le d'$. $\square$

Lemma 9 For a system that is not judged to be schedulable, let $d_m = \max\{d_i \mid 0 < d_i < L\}$. If $h_J(d_m) + B_J(d_m) \le d_m$, when $t \in [h_J(d^\Delta) + B_J(d^\Delta), L)$, $d^\Delta < h_J(t) + B_J(t) \le t$.

Proof. The period $[h_J(d^\Delta)+B_J(d^\Delta), L)$ can be divided into three intervals:

1) $t \in [h_J(d^\Delta)+B_J(d^\Delta), d')$, where $d' = \min\{d_i \mid d_i > d^\Delta\}$, from Lemma 8, $d^\Delta < h_J(d^\Delta)+B_J(d^\Delta) \le d'$, therefore $h_J(t)+B_J(t) = h_J(d^\Delta)+B_J(d^\Delta) \le t$.
2) $t \in [d', d_m)$, then $h_J(t)+B_J(t) = h_J(d_j)+B_J(d_j)$, where $d_j = \max\{d_i \mid d_i \le t\}$. Suppose $t < h_J(t)+B_J(t)$, we have $d_j \le t < h_J(t)+B_J(t) = h_J(d_j)+B_J(d_j)$, since $d^\Delta < d' \le d_j \le L$, this contradicts the condition that d^Δ is the largest d_i satisfying $d_i \in (0, L) \wedge d_i < h_J(d_i)+B_J(d_i)$, therefore $h_J(t)+B_J(t) \le t$.
3) $t \in [d_m, L)$, $h_J(t)+B_J(t) = h_J(d_m)+B_J(d_m) \le d_m \le t$.

Since $h_J(t)+B_J(t)$ is non-decreasing with t, $t \ge h_J(d^\Delta)+B_J(d^\Delta) > d^\Delta$
$\Rightarrow h_J(t)+B_J(t) \ge h_J(d^\Delta)+B_J(d^\Delta) > d^\Delta$, therefore we have $d^\Delta < h_J(t)+B_J(t) \le t$ in each interval. □

Theorem 11 For a general task set scheduled by EDF+SRP with release jitter, let $d_i = kT_i + D_i - J_i, k \in N$, and $d_{\min} = \min_{1 \le i \le n}\{D_i - J_i\}$. If $U \le 1$, we can use the following algorithm to test the schedulability:

$t = \max\{d_i \mid d_i < L\}$;
while $(h_J(t)+B_J(t) \le t \wedge h_J(t)+B_J(t) > d_{\min})$
{***if*** $(h_J(t)+B_J(t) < t)$ $t = h_J(t)+B_J(t)$;
else $t = \max\{d_i \mid d_i < t\}$;
}
if $(h_J(t)+B_J(t) \le d_{\min})$ the task set is schedulable;

Proof. Let $H_B(t)$ be the function $H_B(t) = h_J(t)+B_J(t)$, since Lemma 7 is true, in Theorem 6 & Theorem 7's description and their proof, $h(t)$ can be replaced by $H_B(t)$, $h(L_0)$ can be replaced by $H_B(L_0)$, $h(d_i)$ can be replaced by $H_B(d_i)$, and $h(d_m)$ can be replaced by $H_B(d_m)$. After integrating with blocking factor, Lemma 4 corresponds to Lemma 8, and Lemma 5 corresponds to Lemma 9, so we can use the same process of Theorem 6 and Theorem 7's proof to prove Theorem 11. □

9.3 Illustration Example

We give an example to illustrate the analysis which integrate with both release jitter and blocking. This example contains 6 general tasks, and there are two types of non-preemtable shared resources R1 and R2 in the system. The task parameters and the shared resource

access time as the follows:

Task	Execution Time	Relative Deadline	Period	Release Jitter	R1 Access Time	R2 Access Time
τ_1	7	34	40	6	2	0
τ_2	17	70	136	18	6	9
τ_3	60	280	360	30	22	16
τ_4	49	590	420	40	0	17
τ_5	53	320	510	37	18	12
τ_6	70	360	490	46	16	21

The schedulability is tested by the following steps:

Step 1. Calculate the utilization of the task set, $U = 0.83 \le 1$.

Step 2. Calculate upper bound L_a^B by equation (20), $L_a^B = 509$.

Step 3. Calculate upper bound L_b^J by equation (10), $L_b^J = 766$.

Step 4. As $L_a^B < L_b^J$, $L = L_a^B = 509$. $\max\{d_i \mid d_i < L\} = 478$; $d_{\min} = \min_{1 \le i \le n}\{D_i - J_i\} = 28$.

Verify the schedulability by the QPA algorithm (Theorem 11):

1) $t = 478$, $h_J(t) + B_J(t) = 352$,

2) $t = 352$, $h_J(t) + B_J(t) = 244$,

3) $t = 244$, $h_J(t) + B_J(t) = 98$,

4) $t = 98$, $h_J(t) + B_J(t) = 53$,

5) $t = 53$, $h_J(t) + B_J(t) = 29$,

6) $t = 29$, $h_J(t) + B_J(t) = 22$,

Since $h_J(t) + B_J(t) \le d_{\min}$, the task set is schedulable.

10. Conclusion

In this paper, we have addressed and solved the problem of providing fast schedulability analysis which is necessary and sufficient for EDF scheduling with arbitrary relative deadlines. As the most commonly studied optimal scheduling algorithm, EDF provides real-time systems the best schedulability among all scheduling algorithms. However the large amount of calculation required by its exact schedulability analysis severely restricts the use of EDF in practice.

We present a tighter upper bound for the traditional processor demand analysis which provides exact schedulability test for EDF scheduling, by experimental comparisons, the new upper bound dominants the previous results when each D_i is no larger than $2T_i$. We

propose the Quick convergence Processor Demand Analysis (QPA) which builds on the traditional processor demand analysis, it is necessary and sufficient. By intensive experiments, we show that QPA reduces the required calculation times in logarithm scales in all situations compared with the previous results, and the required calculation by QPA is stable for all kinds of task sets.

Since the QPA algorithm does not check every absolute deadlines, and it does not need all values of deadlines in the time interval, there is no additional calculation overheads needed by QPA, so it can further reduce the calculation in practice beyond the experimental results.

The QPA reduces the required calculation to a level that can be accepted by nearly all computing systems, and it can be easily used for online admission control. By the QPA algorithm, in experiment (G) based on 80,000 and experiment (N) based on 60,000 randomly generated task sets, more than 96% of the task sets complete each schedulability test in less than 30 calculation times of $h(t)$, and all the task sets complete each schedulability test in less than 60 calculations whether they are schedulable or unschedulable. The function $h(t)$ has the complex $O(n)$, equal to calculating a task set's total utilization. This means the vast majority of the task sets in the experiments only require the calculation which is equivalent to less than 30 times the utilization based test.

There is no restriction on the new results: each task can be periodic or sporadic; the relative deadline of each task can be less than, equal to or greater than its period.

We prove that blocking can be considered with release jitter in the traditional processor demand analysis. We also show that QPA can be integrated with both blocking and release jitter, so the general tasks with release jitter can share non-preemptable resources in our system model.

11. Acknowledgements

This work was funded in part by the EU FRESCOR and ARTIST2 projects. The authors would like to thank Robert David for his helpful comments and discussions on the experimental part of this report.

12. References

[1] T.P. Baker. Stack-based Scheduling of Real-Time Processes. *Journal of Real-time Systems*, 3 (1):76-100, 1991.

[2] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. *In Proceedings 11th IEEE Real-Time System Symposium*,

pp.182-190, 1990.
[3] S.K. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor. *Journal of Real-Time Systems*, 4 (2):301-324, 1990.
[4] S.K. Baruah, R.R. Howell, and L.E. Rosier. Feasibility Problems for Recurring Tasks on One Processor. *Theoretical Computer Science*, Vol.118:3-20, 1993.
[5] S.K. Baruah. Resource Sharing in EDF-Scheduled Systems: A Closer Look. *In Proceedings 27th IEEE Real-Time Systems Symposium*, pp.379-387, 2006.
[6] E. Bini and G.C. Buttazzo. Measuring the Performance of Schedulability Tests. *Journal of Real-Time Systems*, 30 (1-2):129-154, 2005.
[7] M.L. Dertouzos. Control robotics:The Procedural Control of Physical Processes. *In Proceedings of the IFIP Congress*, pp.807-813, 1974.
[8] H. Lauer and E. Satterwaite. The Impact of Mesa on System Design. *Proceedings of the 4th International Conference on Software Engineering*, pp.174-182, 1979.
[9] J.Y.-T. Leung and M.L. Merrill. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, pp.115-118, 1980.
[10] C.L. Liu and J.W. Layland. Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment. *Journal of ACM*, 20 (1):40-61, 1973.
[11] J.W.S. Liu. Real-Time Systems. 2000: Prentice-Hall.
[12] I. Ripoll, A. Crespo, and A.K. Mok. Improvement in Feasibility Testing for Real-Time Tasks. *Journal of Real-Time Systems*, 11 (1):19-39, 1996.
[13] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39 (9):1175-1185, 1990.
[14] M. Spuri. Analysis of Deadline Schedule Real-time Systems. *Technical Report 2772, INRIA, France*, 1996.