Schedulability Analysis for Real-Time Systems with EDF Scheduling

Fengxiang Zhang, *Student Member*, *IEEE*, and Alan Burns, *Senior Member*, *IEEE*

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. Existing results on an exact schedulability test for EDF task systems 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 resulting large number of calculations severely restricts the use of EDF in practice. In this paper, we propose new results on necessary and sufficient schedulability analysis for EDF scheduling; the new results reduce, exponentially, the calculation times, in all situations, for schedulable task sets, and in most situations, for unschedulable task sets. For example, a 16-task system that in the previous analysis had to check 858,331 points (deadlines) can, with the new analysis, be checked at just 12 points. There are no restrictions 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, and task parameters can range over many orders of magnitude.

Index Terms—Multiprocessing/multiprogramming/multitasking, scheduling, real-time and embedded systems.

1 Introduction

Real-time systems are playing a crucial role in our society, and in the last two decades, there has been an explosive growth in the number of real-time systems being used in our daily lives and in industry production. Systems such as chemical and nuclear plant control, space missions, flight control systems, military systems, telecommunications, multimedia systems, and so on all make use of real-time technologies. The most important attribute of real-time systems is that the correctness of such systems depends on not only the computed results but also on the time at which results are produced. In other words, real-time systems have timing requirements that must be guaranteed. Scheduling and schedulability analysis enables these guarantees to be provided.

In scheduling theory, a real-time system comprises a set of real-time tasks; each task consists of an infinite or finite stream of jobs. The task set 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 the jobs of all the tasks can be guaranteed to complete their executions before their deadlines. If they can, then we say the task set is schedulable.

Schedulability tests can be sufficient or exact (necessary and sufficient). Sufficient tests are usually efficient but they are not powerful; many schedulable task sets are not judged to be schedulable. The simplest sufficient tests for real-time systems are utilization-based and they have polynomial complexity. However, we observed that nearly all task sets, which are randomly generated in our experiments, cannot be correctly evaluated by such tests [20].

• *The authors are with the RTS Group, Department of Computer Science, University of York, York YO10 5DD, UK.*
E-mail: {zhangfx, burns}@cs.york.ac.uk.

Manuscript received 24 July 2008; revised 14 Jan. 2009; accepted 28 Jan. 2009; published online 24 Mar. 2009.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-07-0367.
Digital Object Identifier no. 10.1109/TC.2009.58.

The exact schedulability analysis for fixed priority (FP) scheduling is accomplished by response time tests [2], [13] which calculate the worst-case response time for each task to judge whether a system is schedulable. However, even with an exact test, FP scheduling is never as effective as the dynamic schemes.

The most common dynamic priority scheduling algorithm for real-time systems is the Earliest Deadline First (EDF) which was introduced by Liu and Layland [15] in 1973. According to the EDF algorithm, an arrived job with the earliest absolute deadline is executed first. The EDF algorithm has been proven by Dertouzos [9] 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 [15] presented a necessary and sufficient schedulability condition for EDF systems under the assumption that all task's relative deadlines are equal to their periods. The schedulability condition is that the total utilization of the task set must be less than or equal to 1. However, in practical real-time systems, a task's relative deadline is not always equal to its period, so the above assumption severely restricts the usefulness of exact utilization-based tests.

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 bounded 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.

0018-9340/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

The motivation for providing faster exact schedulability analysis for general EDF systems is twofold. As part of the design process, many different parameter profiles may need to be checked. An automated search may even be undertaken as part of the architectural definition of the system. An efficient but accurate schedulability scheme is therefore needed. The second requirement comes from online systems. During the runtime of a system, new tasks may arrive that need to be added to the task set. The system must recalculate schedulability online to decide whether to allow the new tasks to enter into the system. Such online admission control gives a much higher requirement for the performance of the schedulability test as the decisions have to be made in a very short time and should not occupy too much system resource.

In this paper, we propose new results on necessary and sufficient schedulability analysis for general EDF systems. We refer to the new approach as the Quick convergence Processor-demand Analysis (QPA) algorithm. QPA builds on the traditional processor demand analysis. By intensive experiments, we show that QPA reduces the calculation effort exponentially in most situations.

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 propose the QPA algorithm which provides fast schedulability tests for arbitrary relative deadline EDF systems. In Section 5, we present a tighter upper bound for the processor demand analysis. Section 6 includes experimental results based on a large number of randomly generated task sets. Conclusions are given in Section 7.

2 System Model

A hard real-time system comprises a set of n independent real-time tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$, each task consists of an infinite or finite stream of jobs or requests which must be completed before their deadlines. Let τ_i indicate 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 arrive at any time at or after $t + T_i$.

It is assumed that the first job of each task arrives at the same time for periodic tasks. 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, where $C_i > 0$, 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 $t + D_i$.

At any time, preemption is allowed. According to the EDF algorithm, an arrived job with an earlier absolute deadline can preempt the execution of a job with a later absolute deadline. When a job completes its execution, the system chooses the pending job with the earliest absolute deadline to execute.

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;
d_i—absolute deadline of a job for task τ_i;
U_i—the utilization of task τ_i, and $U_i = C_i/T_i$;
U—the total utilization of the task set, computed by $U = \sum_{i=1}^{n} C_i/T_i$.

3 Previous Results on Exact Schedulability Analysis

In 1973, Liu and Layland proved that a periodic task set is schedulable if and only if $U \leq 1$ under the assumption that each $D_i = T_i$. Liu [16] reported that the density of a task set given by $\Delta = \sum_{i=1}^{n} C_i / \min\{D_i, T_i\} \leq 1$ is a sufficient schedulability condition for general EDF systems. There are also a number of papers [1], [8], [10] that provide efficient analysis for EDF scheduling, without restriction on relative deadlines, but these tests are only sufficient or are approximations. In the remainder of this section, we concentrate on the exact schedulability analysis for EDF systems with arbitrary relative deadlines.

In 1980, Leung and Merrill [14] noted that a set of periodic tasks is schedulable if and only if all absolute deadlines in the interval $[0, \max\{s_i\} + 2H]$ are met, where s_i is the start time of task $\tau_i, \min\{s_i\} = 0$, and H is the least common multiple of the task periods. In 1990, Baruah et al. [3], [4] extended this condition for sporadic task systems, and they showed that the task set is schedulable if and only if $\forall t > 0, h(t) \leq t$, where $h(t)$ is the processor demand function which calculates the maximum execution time requirement of all tasks' jobs which have both their arrival times and their deadlines in a contiguous interval of length t, and $h(t)$ is given by

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

Baruah et al. [3], [4], [5] showed that using the above necessary and sufficient schedulability test, the value of t can be bound by an easily computed value.

Theorem 1 ([3], [4], [5]). *A general task set (T_i and D_i are not related) is schedulable if and only if $U \leq 1$ and*

$$\forall t < L_a, h(t) \leq t,$$

where L_a is defined as follows:

$$L_a = \max\left\{D_1, \ldots, D_n, \max_{1 \leq i \leq n}\{T_i - D_i\}\frac{U}{1-U}\right\}. \tag{2}$$

In 1996, under the assumption that each $D_i \leq T_i$, Ripoll et al. [18] gave a different upper bound for the schedulability test, their upper bound is

$$L_a^2 = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}. \tag{3}$$

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

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

However, for an arbitrary deadline system in which D_i could be larger than T_i, the maximum relative deadline $\max_{1\le i\le n}\{D_i\}$ has to be taken into account. Therefore, the necessary and sufficient condition for schedulability becomes the following.

Theorem 2 ([7], [11], [12]). *A general task set is schedulable if and only if $U \le 1$ and*

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

where L_a is defined as follows:

$$L_a = \max\left\{D_1, \ldots, D_n, \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}\right\}. \quad (4)$$

In 1996, Spuri [19] and Ripoll et al. [18] derived another upper bound for the time interval which guarantees that 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 described in *Definition 1*). However, Ripoll et al. [18] only considered the situation of each $D_i \le T_i$.

Definition 1 ([18], [19]). *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 [18], [19]:

$$w^0 = \sum_{i=1}^{n} C_i, \quad (5)$$

$$w^{m+1} = \sum_{i=1}^{n} \left\lceil \frac{w^m}{T_i} \right\rceil C_i, \quad (6)$$

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

Lemma 1 ([19]). *The length of the synchronous busy period is the maximum length of any possible busy processor period in any schedule.*

Lemma 2 ([15]). *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 ([19]). *A general task set is schedulable if and only if $U \le 1$ and*

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

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 to be busy at and after $t = L_b$, so the busy period must 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 change at the times of absolute deadlines, the schedulability test becomes the following.

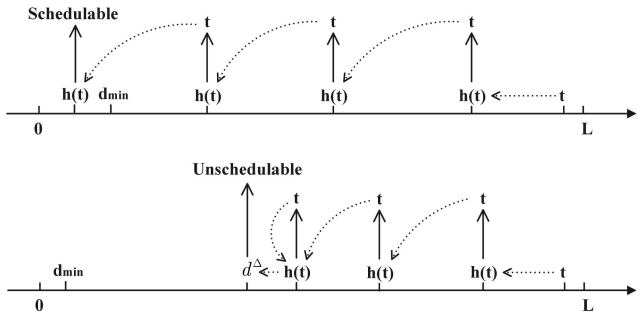

Fig. 1. Illustration of QPA.

Theorem 4 ([5], [7], [11], [12], [19]). *A task set is schedulable if and only if $U \le 1$ and*

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

where $P = \{d_k | d_k = kT_i + D_i \wedge d_k < \min(L_a, L_b), k \in N\}$, where L_a is calculated by (4) and L_b is the solution of (5) and (6).

In Section 5, we will present a minor improvement to the calculation of L_a.

4 New Algorithm for Schedulability Analysis

The existing results on exact schedulability tests for EDF scheduling need to check all the absolute deadlines in the time interval $(0, \min\{L_a, L_b\})$. In a given interval, there can be a very large number of absolute deadlines that need to be checked.

In this section, we propose the QPA algorithm which not only provides fast and simple schedulability tests for EDF, but also is necessary and sufficient. By the proposed algorithm, we do not check every deadline, and we do not need to compute 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 is not well defined (divide by 0) when the utilization of the task U is equal to 1, let L be defined as

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

Let d_i be any absolute deadline of a job from task $\tau_i, d_i = kT_i + D_i, k \in N$. Denote $d_{\min} = \min\{D_i\}$. When a system is unschedulable, define

$$d^{\Delta} = \max\{d_i | 0 < d_i < L \wedge h(d_i) > d_i\}.$$

QPA works by starting with a value of t close to L, and then, iterating back through a simple expression toward 0. The value of this t sequence converges for an unschedulable system to d^{Δ}, and converges for a schedulable system to 0 (although it is stopped once $h(t) \le d_{\min}$, see Fig. 1).

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

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

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

Lemma 5. *For an unschedulable system, let* $d_m = \max\{d_i | d_i < L\}$. *If* $h(d_m) \le d_m$, *then we have* $\forall t \in [h(d^\Delta), d_m), h(d^\Delta) \le h(t) \le t$.

Proof. From Lemma 4, $d^\Delta < h(d^\Delta) < d'$, so when $t \in [h(d^\Delta), d'), h(t) = h(d^\Delta) \le t$. When $t \in [d', d_m)$, suppose $\exists t_\kappa < h(t_\kappa)$, let $d_\kappa = \max\{d_i | d_i \le t_\kappa\}$, then $h(d_\kappa) = h(t_\kappa) > t_\kappa \ge d_\kappa$, since $d_\kappa > t^\Delta$. This contradicts the definition of d^Δ, so there is no such t_κ in $[d', d_m)$. Therefore, $\forall t \in [h(d^\Delta), d_m), h(d^\Delta) \le h(t) \le t$. □

Theorem 5. *A general task set is schedulable if and only if* $U \le 1$ *and the result of the following iterative algorithm is* $h(t) \le d_{\min}$.

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

Proof. Suppose the task set is not schedulable. Again, let $d_m = \max\{d_i | d_i < L\}$.

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 before the iterative process stops.

Case 1: $h(t) < t$. At the beginning of the iteration, $t = d_m > d^\Delta$. Since $h(t)$ is a nondecreasing function with $t, h(t) \ge h(d^\Delta)$, then we have $h(d^\Delta) \le h(t) < t < d_m$, from Lemma 5, $h(d_m) \le d_m \wedge t \in [h(d^\Delta), d_m] \Rightarrow h(d^\Delta) \le h(t) \le t$, so after $t \leftarrow h(t)$, we still have $h(d^\Delta) \le h(t) < t < d_m$. Therefore, t is always larger than or equal to $h(d^\Delta)$ in this case.

Case 2: $h(t) = t$. At this time, t is still larger than d^Δ, and if we let $t \leftarrow \max\{d_i | d_i < t\}$, obviously $t = d_i \ge d^\Delta$. If $t = d_i = d^\Delta$, then $h(t) > t$, and the iterative process stops. If $t = d_i > d^\Delta$, from Lemma 4, $d^\Delta < h(d^\Delta) < d'$, where $d' = \min\{d_i | d_i > d^\Delta\}$; therefore, we have $h(d^\Delta) < d' \le t < d_m$, and from Lemma 5, the process enters Case 1 or Case 2 again.

Case 3: $t = h(d^\Delta)$. From Lemma 4, $d^\Delta < h(d^\Delta) < d'$, so $h(t) = h(d^\Delta) = t$, that is, an example of Case 2, and if we let $t \leftarrow \max\{d_i | d_i < t\}$, 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 larger than or equal to d^Δ (stopped with $t = d^\Delta$), and we have $h(t) \ge h(d^\Delta) > d^\Delta \ge d_{\min}$. Therefore, when $h(t) \le d_{\min}$, the task set is schedulable. □

Theorem 5 is also a process to find d^Δ for unschedulable task sets; hence, it can also be presented as: a general task set is schedulable if and only if $U \le 1$ and the algorithm cannot find d^Δ. For a schedulable system, when $h(t) \le d_{\min}$, if we change the stopping condition to let the iterative process continue, then after one or two more iterations, t will converge at 0.

In the iterative process of Theorem 5, t takes the value $h(t)$, and when $h(t) < t$ progress toward zero is made. Only when $h(t) = t$, we need to force the process to take a value less than $h(t)$. This is when we need to compute $\max\{d_i | d_i < t\}$ to let the iteration continue; $\max\{d_i | d_i < t\}$ can be calculated by the following approach.

For a single task τ_j with $D_j < t$, the last arrived job of task τ_j with $d_j \le t$ is released at:

$$\left\lfloor \frac{t - D_j}{T_j} \right\rfloor T_j,$$

and the absolute deadline of this job is:

$$d_j = \left\lfloor \frac{t - D_j}{T_j} \right\rfloor T_j + D_j. \tag{8}$$

If $d_j = t$, we let d_j move to the previous deadline $d_j = d_j - T_j$. For the task set, $\max\{d_i | d_i \le t\}$ is the largest such d_j for each task.

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

for $(j = 1; j \le n; j++)$
 $\{$***if*** $(D_j < t)$
 $\{d_j \leftarrow \lfloor (t - D_j)/T_j \rfloor T_j + D_j$;
 if $(d_j = t)\ d_j \leftarrow d_j - T_j$;
 if $(d_j > d^t_{\max})\ d^t_{\max} \leftarrow d_j$;
$\}\}$
After the recurrence, $d^t_{\max} = \max\{d_i | d_i \le t\}$.

The above algorithm for finding the $\max\{d_i | d_i \le t\}$ has the complexity $O(n)$, which is only equivalent to one $h(t)$ calculation.

Illustration of QPA. The following example illustrates the QPA test and its improvement. The task set includes eight 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

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 (4), $L_a = 18{,}000$. (There are 1,735 absolute deadlines in $(0, L_a)$.)

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

Step 3. Calculate upper bound L_b by (5) and (6), $L_b = 16{,}984$. (There are 1,638 absolute deadlines in $(0, L_b)$.)

(Using the minimum value mean, there are 1,638 absolute deadlines that must be checked by the old approach.)

Step 4. $L_b < L_a, L = L_b = 16{,}984$; $d_{\min} = 16$ and $\max\{d_i | d_i < L\} = 16{,}974$. Check the schedulability by the QPA algorithm given in Theorem 5:

1. $t = 16{,}974, h(t) = 8{,}890$,
2. $t = 8{,}890, h(t) = 3{,}080$,
3. $t = 3{,}080, h(t) = 1{,}098$,
4. $t = 1{,}098, h(t) = 362$,
5. $t = 362, h(t) = 118$,
6. $t = 118, h(t) = 26$,
7. $t = 26, h(t) = 2$.

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

Only 7 iterations required compared to 1,638 previously. More illustration examples can be found in [20].

5 Improvement to the Upper Bound L_a

In this section, based on Theorem 1, we give a tighter upper bound for the exact schedulability test; there is no restriction on the relationship between relative deadlines and periods.

Theorem 6. *A general task set is schedulable if and only if* $U \leq 1$ *and*

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

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

$$L_a^* = \max\left\{(D_1 - T_1), \ldots, (D_n - T_n), \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1 - U}\right\}. \tag{9}$$

Proof. When $t \geq \max_{1 \leq i \leq n}\{D_i - T_i\} \Leftrightarrow t \geq D_i - T_i \Leftrightarrow t - D_i \geq -T_i \Leftrightarrow \lfloor \frac{t - D_i}{T_i} \rfloor \geq -1 \Leftrightarrow 1 + \lfloor \frac{t - D_i}{T_i} \rfloor \geq 0$, then we have

$$\begin{aligned} h(t) &= \sum_{i=1}^{n} \max\left\{0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right\} C_i \\ &= \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i, \\ &\leq t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i). \end{aligned}$$

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

$$\begin{aligned} &\Rightarrow \quad t < t \sum_{i=1}^{n} \frac{C_i}{T_i} + \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i), \\ &\Leftrightarrow \quad t\left(1 - \sum_{i=1}^{n} \frac{C_i}{T_i}\right) < \sum_{i=1}^{n} \frac{C_i}{T_i}(T_i - D_i), \\ &\Leftrightarrow \quad t < \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1 - U}. \end{aligned}$$

□

To incorporate the upper bound L_a^* into QPA, in Section 4, we only need to let $L = \min(L_a^*, L_b)$ when $U < 1$.

6 Experiments and Evaluations

This section describes experiments that have been conducted to evaluate the performance of the QPA algorithm. We compare the number of calculations required by the original approach with upper bounds L_a, L_b, and L_a^*, and the QPA algorithm. Comparison is by an extensive number of experiments on a large range of task sets with randomly generated parameters.

Since the task generation policies can significantly affect experimental results, we give the details of the policies we used in the experiments as follows.

Utilizations generation policy. In order to get uniformly 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 range of task periods is significant. If we want to explore say six orders of magnitude (e.g., 1-1,000,000), then a set of random choices within this range will result in 99 percent of values being in the range 10,000-1,000,000 (only three orders of magnitude are actually explored in the expected six orders). Hence, we use the approach recommended by Davis and Burns [17] to generate the task periods according to an exponential distribution. Let $T_{\max} = \max_{1 \leq i \leq n}\{T_i\}$, and $T_{\min} = \min_{1 \leq i \leq n}\{T_i\}$. In order to make sure the periods are uniformly distributed in a given range (the maximum value of $T_{\max}/T_{\min}$), the range of the periods are divided into the intervals $e^o \sim e^1, e^1 \sim e^2$, $e^2 \sim e^3, \ldots$; if there are k intervals, then $\lfloor (n-1)/k \rfloor$ task periods are generated randomly in each interval, and the remaining $((n-1) \bmod k)$ task periods are generated randomly in intervals $e^o \sim e^1$ through $e^{((n-1) \bmod k)-1} \sim e^{(n-1) \bmod k}$. In each interval, the task periods are generated according to a uniform distribution, by using the rand() function of the C language.

For example, if the number of tasks is 14, and we want $T_{\max}/T_{\min} \leq 100$, first let $T_{14} = 100$, and then 13 task periods are generated between 1 and T_{14}. Since $\ln 100 \cong 4.605$, the range is divided into five intervals, which are $e^o \sim e^1$, $e^1 \sim e^2, \ldots, e^4 \sim e^{4.605}$. Then, two task periods are generated randomly in each interval, and for the other three periods, one is generated in each of three randomly chosen intervals, shown as follows:

$$\begin{aligned} &e^o \sim e^1 : \tau_1, \tau_2, \tau_{11}, \\ &e^1 \sim e^2 : \tau_3, \tau_4, \tau_{12}, \\ &e^2 \sim^3 : \tau_5, \tau_6, \tau_{13}, \\ &e^3 \sim^4 : \tau_7, \tau_8, \\ &e^4 \sim^{4.605} : \tau_9, \tau_{10}. \end{aligned}$$

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 $10 \leq C_i < 100, a = 2 \times C_i$; when $100 \leq C_i < 1{,}000, a = 3 \times C_i$; and when $C_i \geq 1{,}000$, $a = 4 \times C_i$. The default value $b = 1.2 \times T_i$.

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

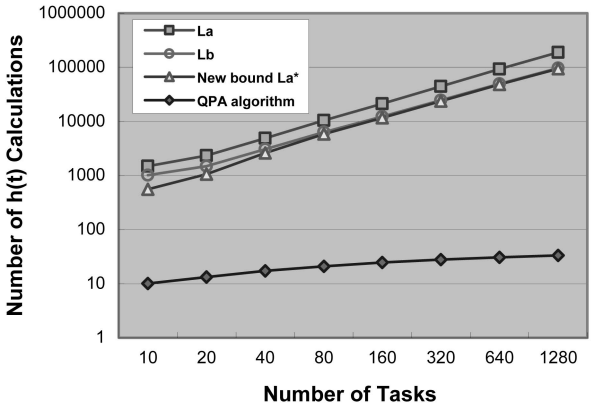

Fig. 2. Impact of the number of tasks.

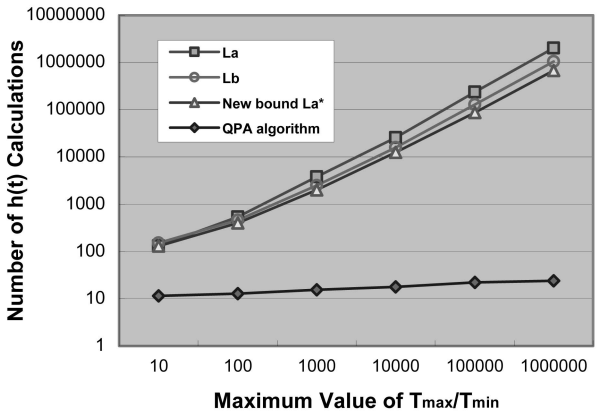

Fig. 3. Impact of the task periods range.

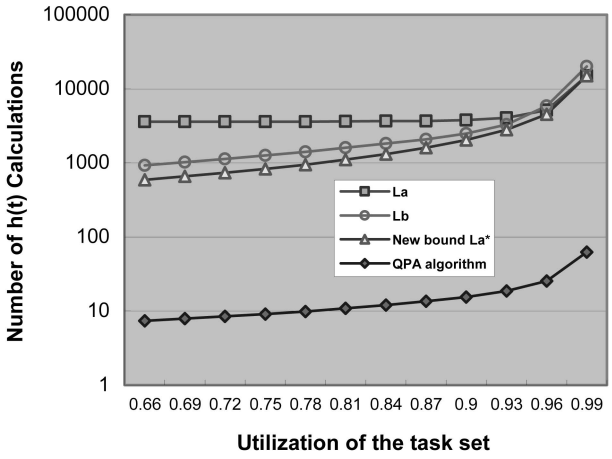

Fig. 4. Impact of the task set's utilization.

In some experiments, we 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.

Measurement metric. 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 approach needs to check all the absolute deadlines in the upper bound when a task set is schedulable (but can stop once a deadline miss is found for an unschedulable system), we do the experiments separately for schedulable and unschedulable task sets. When we experiment on schedulable task sets, if a generated task set is unschedulable, 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).

6.1 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. Each point on the diagrams is the average of 6,000 randomly generated schedulable task sets.

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

Since the density of a task set:

$$\Delta = \sum_{i=1}^{n} C_i / \min\{D_i, T_i\} \leq 1$$

is a utilization-based condition which is sufficient for EDF schedulability, in each experiment of this section, we also tested the density of each schedulable task set to see what percentage of the task sets have $\Delta > 1$, in which case we cannot judge feasibility by this simple but sufficient test. In these experiments, we observed that nearly all the schedulable task sets have the density $\Delta > 1$, and hence, an exact schedulability test is required.

6.1.1 Impact of the Number of Tasks

In this experiment, we let each task set's utilization be 0.9 and the maximum value of $T_{\max}/T_{\min}$ be 1,000. The number of the required calculation times is therefore a function of the number of tasks. Results of this experiment are illustrated in Fig. 2.

6.1.2 Impact of the Task Periods Range

In this experiment, the number of tasks for each task set is 30, and each task set's utilization is 0.9. The number of calculation times is thus a function of the maximum value of $T_{\max}/T_{\min}$. Results of this experiment are illustrated in Fig. 3.

6.1.3 Impact of the Utilization

This experiment investigates the impact of the task set's utilization. The size of each task set is 30 and $T_{\max}/T_{\min} \leq 1{,}000$. Results of this experiment are illustrated in Fig. 4.

6.1.4 Impact of the Maximum Value D_i/T_i

Here, 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 $b = \max\{D_i/T_i\} \times T_i$. The utilization of each task set is 0.9, task number is 30, and

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

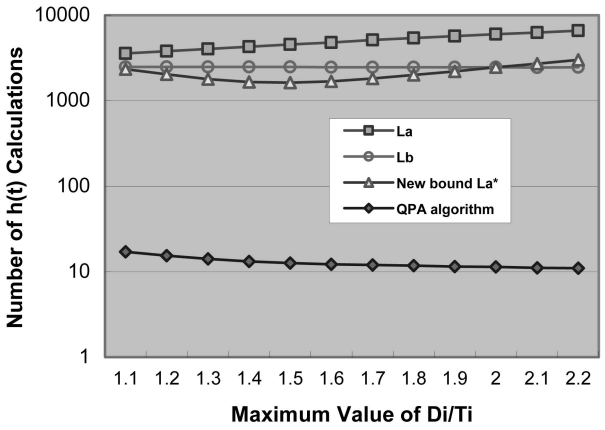

Fig. 5. Impact of the maximum value of D_i/T_i.

$T_{\max}/T_{\min} \leq 1{,}000$. The number of calculation times is therefore a function of the value $\max\{Di/Ti\}$. Results of this experiment are illustrated in Fig. 5.

Our default generation policy for each relative deadline is $b = \max\{D_i/T_i\} = 1.2$. From the above experiment, we can see that the QPA algorithm can perform faster when $b > 1.2$.

6.1.5 Frequency Distribution of the Task Sets

This experiment explores what percentage of the task sets can complete each schedulability test in a given number of $h(t)$ calculations by QPA. We divide each interval on the x-axis into 10; this means if a schedulable task set needs 12 calculations to complete its schedulability test, then this task set is counted into the interval $10 \sim 20$. The value of the y-axis presents the percentage of the tested task sets.

This experiment is based on 80,000 randomly generated task sets which are schedulable. 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 10,000. Results of this experiment are illustrated in Fig. 6.

We can see from Fig. 6 that the vast majority of task sets complete each schedulability test in less than 30 calculations of $h(t)$, and all the 80,000 task sets in our experiment complete each schedulability test in less than 60 times calculations.

6.2 Experiments on Unschedulable Task Sets

As QPA starts from L and works backward toward 0, it would seem to be at a disadvantage, as the first deadline miss is often closer to 0. Indeed, it is possible to construct example task sets that fail on the very first deadline but which take QPA 30 or more interactions before concluding that the task set is unschedulable. In addition, the simulation results [20] show that checking forward from $t = 0$ using a presorted sequence of all absolute deadlines finds an overflow with the least number of checked deadlines. However, sorting incurs significant extra efforts. Nevertheless, we will show in the following experiments that, overall, QPA does perform significantly better than the previous approaches with the forward order, even when the cost of sorting (which is required by the old method) is ignored—as it is in these experiments.

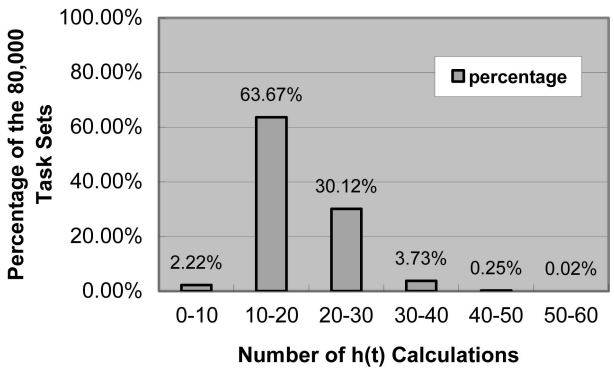

Fig. 6. Frequency distribution of QPA based on 80,000 schedulable task sets.

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

Each point on the diagram is the average of 6,000 randomly generated unschedulable task sets. The "old method" on the graph means the number of absolute deadlines which have been checked by the original approach using the forward-order scheme. Note that these results are presented using a linear y-axis.

6.2.1 Impact of the Task Periods Range

In this experiment, the utilization of each task set is 0.9 and the size of each task set is 30. Results of this experiment are illustrated in Fig. 7.

The default value of the maximum value of $T_{\max}/T_{\min}$ for all following experiments in this section is set to 1,000. From experiments described in Sections 6.1.2 and 6.2.1, we can see that if the maximum value of $T_{\max}/T_{\min}$ is larger than 1,000, then the improvement of QPA over the old approach will increase significantly.

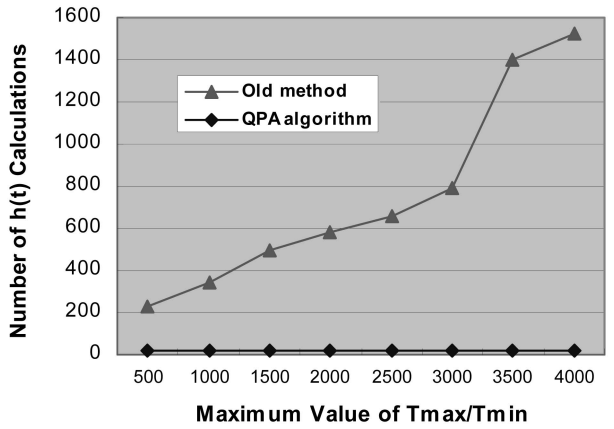

Fig. 7. Impact of the task periods range.

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

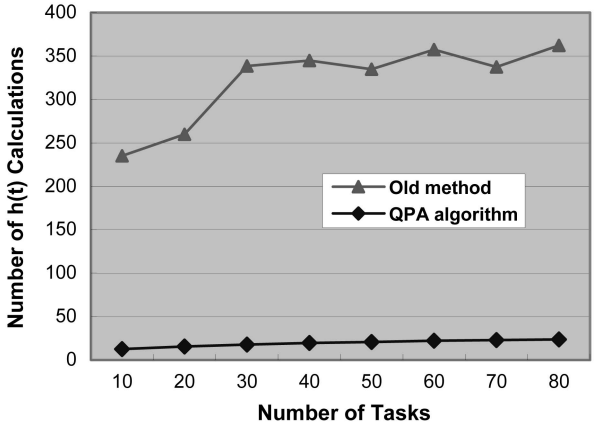

Fig. 8. Impact of the number of tasks.

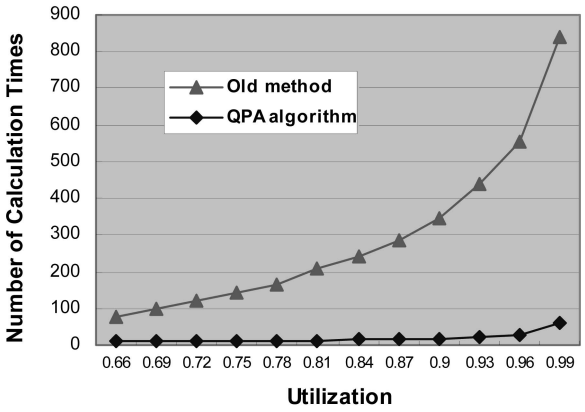

Fig. 9. Impact of the task set's utilization.

6.2.2 Impact of the Tasks Number

In this experiment, the utilization of each task set is 0.9 and the maximum value of $T_{\max}/T_{\min}$ is 1,000. Results of this experiment are illustrated in Fig. 8.

6.2.3 Impact of the Task Set's Utilization

In this experiment, we let the size of each task set be 30 and the maximum value of $T_{\max}/T_{\min}$ be 1,000. Results of this experiment are illustrated in Fig. 9.

6.2.4 Frequency Distribution of the Task Sets

This experiment is 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 1,000. Results of this experiment are illustrated in Fig. 10.

7 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. From the experiments described in Section 6 and a more extensive range of experiments recorded in an available technical report [20], we can see that a number of factors can significantly affect the experimental results of the old methods; in some circumstances, they have exponential

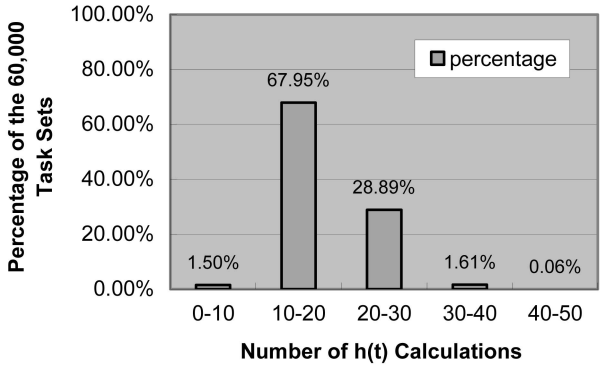

Fig. 10. Frequency distribution of QPA based on 60,000 unschedulable task sets.

growth. The experimental results for QPA are similar for all kinds of task sets, and QPA reduces the required number of calculations exponentially in almost all situations.

Across all experiments (i.e., 80,000 schedulable and 60,000 unschedulable task sets), we observed that by QPA, more than 96 percent of the task sets complete each schedulability test in less than 30 calculations of $h(t)$. The function $h(t)$ has the complexity $O(n)$, equal to calculating a task set's utilization. This means that the vast majority of the task sets in the experiments only require a calculation which is equivalent to less than 30 times the utilization-based test. The previous methods (average across all experiments) require 65,000 such tests.

We also observed that the new upper bound L_a^* dominates L_b when each D_i is no larger than $2T_i$. The calculation of L_b requires an iterative process which may need more iterations than the entire QPA algorithm. Since L_a^* is simpler to calculate than L_b, we would suggest that only L_a^* is used in QPA when each D_i is not significantly larger than $2T_i$.

Acknowledgments

This work was funded in part by the EU FRESCOR and ARTIST projects. The authors would like to thank Rob Davis for his helpful comments and discussions on the experimental portion of this paper.

References

[1] K. Albers and F. Slomka, "An Event Stream Driven Approximation for the Analysis of Real-Time Systems," *Proc. 16th Euromicro Conf. Real-Time Systems*, pp. 187-195, 2004.

[2] N.C. Audsley, A. Burns, M. Richardson, K.W. Tindell, and A.J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling," *Software Eng. J.*, vol. 8, no. 5, pp. 284-292, 1993.

[3] S.K. Baruah, A.K. Mok, and L.E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," *Proc. 11th IEEE Real-Time System Symp.*, pp. 182-190, 1990.

[4] 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," *J. Real-Time Systems*, vol. 4, no. 2, pp. 301-324, 1990.

[5] S.K. Baruah, R.R. Howell, and L.E. Rosier, "Feasibility Problems for Recurring Tasks on One Processor," *Theoretical Computer Science*, vol. 118, pp. 3-20, 1993.

[6] E. Bini and G.C. Buttazzo, "Measuring the Performance of Schedulability Tests," *J. Real-Time Systems*, vol. 30, nos. 1/2, pp. 129-154, 2005.

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.

[7] G.C. Buttazzo, "Real-Time Scheduling and Resource Management," *Handbook of Real-Time and Embedded Systems.* Chapman & Hall/CRC, 2008.
[8] S. Chakraborty, S. Kunzli, and L. Thiele, "Approximate Schedulability Analysis," *Proc. IEEE Real-Time Systems Symp.*, pp. 159-168, 2002.
[9] M.L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes," *Proc. Int'l Federation for Information Processing (IFIP) Congress,* pp. 807-813, 1974.
[10] M. Devi, "An Improved Schedulability Test for Uniprocessor Periodic Task Systems," *Proc. 15th Euromicro Conf. Real-Time Systems,* pp. 23-30, 2003.
[11] L. George, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling," Technical Report 2966, INRIA, 1996.
[12] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson, "Computing the Minimum EDF Feasible Deadline in Periodic Systems," *Proc. 12th IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications,* pp. 125-134, 2006.
[13] M. Joseph and P.K. Pandya, "Finding Response Times in a Real-Time System," *The Computer J.,* vol. 29, no. 5, pp. 390-395, 1986.
[14] J.Y.-T. Leung and M.L. Merrill, "A Note on Preemptive Scheduling of Periodic, Real-Time Tasks," *Information Processing Letters,* vol. 11, no. 3, pp. 115-118, 1980.
[15] C.L. Liu and J.W. Layland, "Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 40-61, 1973.
[16] J.W.S. Liu, *Real-Time Systems.* Prentice-Hall, 2000.
[17] R.I. Davis and A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems," *IEEE Trans. Computers,* vol. 57, no. 9, pp. 1261-1271, Apr. 2008.
[18] I. Ripoll, A. Crespo, and A.K. Mok, "Improvement in Feasibility Testing for Real-Time Tasks," *J. Real-Time Systems,* vol. 11, no. 1, pp. 19-39, 1996.
[19] M. Spuri, "Analysis of Deadline Schedule Real-Time Systems," Technical Report 2772, INRIA, 1996.
[20] F. Zhang and A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling," Technical Report YCS-426-2008, Dept. of Computer Science, Univ. of York, 2008.

Fengxiang Zhang received the BSc degree in information and computing science from Chongqing University of Posts and Telecommunications, China, in 2003. He is currently working toward the PhD degree in the Computer Science Department at the University of York in UK. From 2003 to 2006, he was a teaching assistant in the School of Computer Science, Southwest University, China. His current research interests are in the general area of real-time scheduling analysis, including scheduling algorithms, schedulability analysis, hierarchical scheduling, and multiprocessor real-time systems. He is a student member of the IEEE.

Alan Burns is a professor of real-time systems in the Department of Computer Science, University of York, UK. His research interests cover a number of aspects of real-time systems, including the assessment of languages for use in the real-time domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable multifaceted real-time applications. He has authored/coauthored 400 papers/reports and 15 books. Many of these are in the real-time area. His teaching activities include courses in Operating Systems and Real-time Systems. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY. Downloaded on October 09,2020 at 01:29:08 UTC from IEEE Xplore. Restrictions apply.