Improvement to Quick Processor-demand Analysis for EDF-Scheduled Real-Time Systems

Fengxiang Zhang and Alan Burns

Real-Time Systems Research Group,

Department of Computer Science, University of York, UK

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

Abstract

Earliest Deadline First (EDF) is an optimal scheduling algorithm for uniprocessor real-time systems. Quick Processor-demand Analysis (QPA) provides efficient and exact schedulability tests for EDF scheduling with arbitrary relative deadline. In this paper, we propose Improved Quick Processor-demand Analysis (QPA) which is based on QPA. By extensive experiments, we show that QPA* can significantly reduce the required calculations to perform an exact test for unschedulable systems. We prove that the computation time for testing schedulable systems is hardly affected. Hence the required calculations for general systems can be significantly decreased.*

1. Introduction

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. Real-time systems have to guarantee that all the strict timing requirements must be satisfied. In other words, real-time systems have timing requirements that must be guaranteed. Scheduling and schedulability analysis enables these guarantees to be provided.

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 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 timing deadlines. If they can then we say the task set is schedulable.

The most common dynamic scheduling scheme for real-time systems is Earliest Deadline First (EDF) which was introduced by Liu and Layland [10] in 1973. According to the EDF algorithm, an arrived job with the earliest absolute deadline is executed first. The EDF algorithm has been proven [6] 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.

Schedulability tests for general EDF systems with arbitrary relative deadlines 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 are utilization based and they have polynomial complexity, however we observed that nearly all the task sets which are randomly generated in our experiments [14] cannot be correctly evaluated by such tests. Exact tests can be performed by processor demand analysis, which calculates the processor demand of a task set at every absolute deadline to check if there is an overflow in a specified time interval. In such an interval, there could be a very large number of deadlines that need to be verified. The significant effort required to perform the exact test severely restricts the use of EDF in practice.

Quick Processor-demand Analysis (QPA) [16] provides fast and exact schedulability tests for general EDF systems. By extensive experiments [14, 16], we showed that QPA decreases the required calculations exponentially compared with previous results on exact schedulability tests; the required calculation load for QPA is stable for all kinds of task sets.

The motivation for providing faster exact schedulability analysis for general EDF systems is two-fold. 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 run-time of a system there could be new tasks arrive that need (if possible) 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 Improved Quick Processor-demand Analysis (QPA*) which is based on QPA. We show that QPA* can significantly reduce the required calculations to perform an exact test for unschedulable systems, and we prove that the computation required by QPA* for schedulable systems is nearly the same as QPA. Hence the required calculations for general EDF systems can be significantly decreased by QPA*.

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 analysis for EDF systems with arbitrary relative deadlines. Section 4 describes QPA which was proposed in [16]. Section 5 gives the motivation to the improvement. In section 6, we propose the QPA* algorithm, and prove some properties of QPA*. In Section 7, by intensive experiments on a large number of randomly generated task sets, we show that QPA* can significantly reduce the required calculations for an exact schedulability test. Conclusions are provided in Section 8.

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 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 arrival at any time at or after $t + T_i$.

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 first job of each periodic task is assumed to 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 $t + D_i$.

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

d_i —an absolute deadline of a job of 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, and $U = \sum_{i=1}^{n} C_i / T_i$.

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 (i.e. D_i not necessarily equal to T_i). 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. [1, 2] 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. [1, 2, 3] showed that using the above necessary and sufficient schedulability test, the value of t can be bound by a certain value.

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

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

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 $D_i \le T_i$ for all tasks, the upper bound is:

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

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

$$\le \max_{1 \le i \le n}\{T_i - D_i\} \frac{U}{1-U}$$

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

Theorem 2 [5, 7, 8] 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:

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

In 1996, Spuri [13] 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, 13] 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, 13]:

$$w^0 = \sum_{i=1}^{n} C_i, \quad (4)$$

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

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

Lemma 1 [13] The length of the synchronous busy period is the maximum length of any possible busy processor period in any schedule.

Theorem 3 [13] 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 2 [16] $h(L_b) \le L_b$.

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 of the system's jobs, the schedulability test becomes:

Theorem 4 [3, 5, 7, 8, 13] A task set is schedulable if and only if $U \le 1$ and

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

where

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

Theorem 5 [16] A general task set is schedulable if and only if $U \le 1$ and

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

where

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

and where

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

Clearly $L_a^* < L_a$. Also in extensive simulation studies [14] it was nearly always the case that $L_a^* < L_b$.

In a given interval (i.e. between 0 and $\min\{L_a^*, L_b\}$), there can be a very large number of absolute deadlines that need to be checked. This level of computation has been a serious disincentive to the adoption of EDF scheduling in practice. Fortunately, a new much less intensive test known as Quick convergence Processor-demand Analysis (QPA) has been proposed [16]. QPA works by starting with a value of t close to L and then iterating back through a simple expression toward 0 or the largest failure point in $(0, L)$. It jumps over deadlines that can safely be ignored and hence only a small number of points require to be checked.

4. Quick convergence Processor-demand Analysis (QPA)

This section describes some work that has been reported in [14, 16], as a foundation of Section 5.

Let d_i be an absolute deadline of a job for task τ_i, then $d_i = kT_i + D_i$ for some k, $k \in N$. Let L 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:

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

when a system is unschedulable, define d^Δ to be the largest 'failing' deadline less than L, that is:

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

Lemma 3 [16] For a unschedulable system, let $d_m = \max\{d_i \mid d_i < L\}$. If $h(d_m) \le d_m$, then: $d^\Delta < h(d^\Delta) < d'$, where d' is the closest deadline of d^Δ, and $d' = \min\{d_i \mid d_i > d^{\Delta}\}$.

Lemma 4 [16] For a unschedulable system, let $d_m = \max\{d_i \mid d_i < L\}$. If $h(d_m) \le d_m$, then we have $\forall t \in [h(d^{\Delta}), d_m)$, $h(d^{\Delta}) \le h(t) \le t$.

Theorem 6 [16] A general task set is schedulable if and only if $U \le 1$, and the result of the following iterative algorithm is $h(t) \le d_{\min}$, where $d_{\min} = \min\{D_i\}$.

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

In the iterative process of Theorem 6, 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 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 with $D_j < t$, the last arrived job of task τ_j with $d_j \le t$ is released at:

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

the absolute deadline of this job is:

$$d_j = \left\lfloor \frac{t - D_j}{T_j} \right\rfloor T_j + D_j. \qquad (7)$$

If $d_j = t$, we let d_j move to the previous deadline $d_j = d_j - T_j$. For the task set, $\max\{d_i \mid d_i \le t\}$ 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* ( $j = 1$; $j \le n$; $j{+}{+}$ )
  {*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, $d^t_{\max} = \max\{d_i \mid d_i \le t\}$.

The following graph shows an experimental comparison between processor demand analysis (PDA) and QPA. Each point on the diagram is the average of 6,000 randomly generated schedulable task sets, for each task set $U = 0.9$ and the periods range is $1 \sim 1000$.

The figure is derived from the results presented in [14, 16] where the details of the experiments are given. Note the logarithmic scale in the diagram.

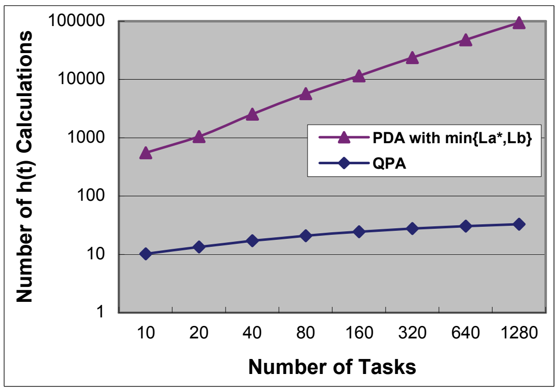

Figure 1. Improvement of QPA

An alternative algorithm which further improves QPA is provided in the appendix of this paper.

5. QPA Test for Unschedulable Systems

QPA is a process for finding the largest failure point for a unschedulable system. Here we give an example to illustrate this.

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	10	96
τ_6	2	16	12
τ_7	10	19	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 (6), $L_a^* = 15404$.

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

As $L_a^* < L_b$, $L = L_a^* = 15404$; $d_{\min}$ =10 and $\max\{d_i \mid d_i < L\}$=15400.

Step 4. Verify schedulability by the QPA algorithm given in Theorem 6:

1) $t = 15400$, $h(t)$=8298,
2) $t = 8298$, $h(t)$=2896,
3) $t = 2896$, $h(t)$=970,

4) $t = 970$, $h(t)$=340,
5) $t = 340$, $h(t)$=134,
6) $t = 134$, $h(t)$=46,
7) $t = 46$, $h(t)$=24;
8) $t = 24$, $h(t)$=20;
9) $t = 20$, $h(t)$=20;
10) $t = 19$, $h(t)$=20;

Since $h(t) > t$, the task set is unschedulable.

In the above example, the largest failure point in $(0, L)$ (i.e. (0,15404)) is $t = 19$ which is very close to 0.

The following experiment investigates the distance from the largest failure point to 0 based on 10,000 unschedulable task sets which are randomly generated.

Let d_x be the largest failure point where the QPA test stops, and

$$d_x = \max\{d_i \mid 0 < d_i < L \wedge h(d_i) > d_i\}.$$

Then the value of d_x / L presents the position of d_x in $(0, L)$.

From Figure 2, we can see that the values of most d_xs are close to 0.

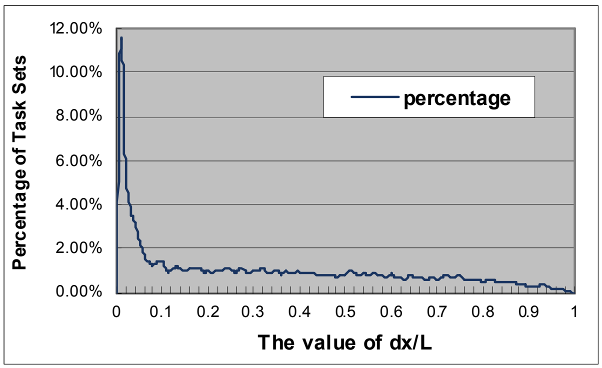

Figure 2. Frequency distribution of d_x based on 10,000 unschedulable task sets, for each task set, U=0.9, n=30 and Tmax/Tmin=1000.

6. Improvement to Quick Processor-demand Analysis (QPA*)

Motivated by the observations of the previous section, we divide the interval $(0, L)$ into two subintervals $(0, L_x)$ and (L_x, L) where L_x is much closer to 0 than L. QPA* starts by checking the first interval, if a failure is found the system is unschedulable. If no failure is found in the first interval then the second is checked. In both intervals the algorithm starts with the value t ($t = \max\{d_i \mid d_i < L_x$ or $d_i < L$ }). A failure may be found in the second interval or the schedulability of system may be proven. Figure 3 illustrates two of the possible situations.

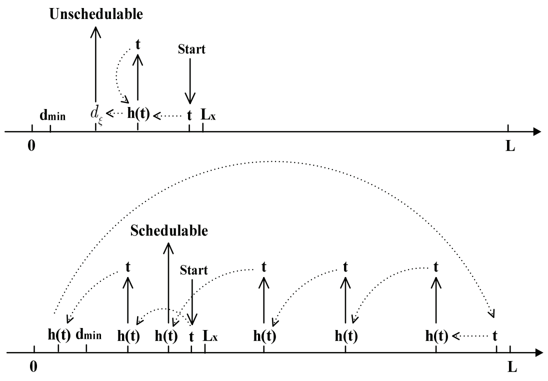

Figure 3. QPA* with one Lx

Theorem 7 Let $L_x \in (0, L)$. A general task set is schedulable if and only if $U \le 1$, and the result of the following algorithm is $find$ = false.

```
$find \leftarrow$ false;
$t \leftarrow \max\{d_i \mid d_i < L_x\}$;
***while*** ( $h(t) > d_{\min} \wedge find$ = false)
 { ***if*** ( $h(t) < t$ )  $t \leftarrow h(t)$;
   ***else if*** ( $h(t) > t$ )  $find \leftarrow$ true;
   ***else***  $t \leftarrow \max\{d_i \mid d_i < t\}$;
 }
***if*** ( $find$ = false)
 { $t \leftarrow \max\{d_i \mid d_i < L\}$;
   ***while*** ( $h(t) > L_x \wedge find$ = false)
    { ***if*** ( $h(t) < t$ )  $t \leftarrow h(t)$;
      ***else if*** ( $h(t) > t$ )  $find \leftarrow$ true;
      ***else***  $t \leftarrow \max\{d_i \mid d_i < t\}$;
    }
 }
***if*** ( $find$ = false)   the system is schedulable;
***else***   the system is unschedulable;
```

Proof. Let $d_\lambda = \max\{d_i \mid 0 < d_i < L_x \wedge h(d_i) > d_i\}$ if there d_λ exists. Lemma 3 and Lemma 4 still hold when L is substitute by L_x, then we can apply the same discussion of Theorem 6's proof [16] to prove that during the iterative process, the value of t is always greater than or equal to d_λ, and $h(t) \ge h(d_\lambda) > d_\lambda \ge d_{\min}$. So when $h(t) \le d_{\min}$, there is no failure point in $(0, L_x)$.

Let $d_\gamma = \max\{d_i \mid L_x \le d_i < L \wedge h(d_i) > d_i\}$ if it exists, after $t \leftarrow \max\{d_i \mid d_i < L\}$, the value of t is always greater than or equal to d_γ, and we have $h(t) \ge h(d_\gamma) > d_\gamma \ge L_x$. Therefore when $h(t) \le L_x$, there is no failure point in $[L_x, L)$.

From the above discussion, if no missed deadline can be found during the whole iterative process, the task set is schedulable, otherwise it is unschedulable. □

As the largest failure point in the interval $(0,L)$ is often close to 0, the QPA* algorithm which is described in Theorem 7 can reduce the calculations for unschedulable systems. For a schedulable system, since QPA* divides $(0,L)$ into two intervals, each interval has to be checked, so the required calculations by QPA* could be more than QPA. However we can prove that for a schedulable system, the required calculation by QPA* is nearly the same as QPA, and at most one more $h(t)$ calculation is required for the whole test.

Lemma 5 During the iterations of QPA or QPA*, when $h(t)=t$, if t is not an absolute deadline of any task, then the system is unschedulable.

Proof. Let $t_\kappa = h(t_\kappa)$, and $t_\kappa \neq d_i$. Let $d_\psi = \max\{d_i \mid d_i < t_\kappa\}$, then $h(d_\psi) = h(t_\kappa) > d_\psi$, the system is unschedulable. □

Lemma 5 can be used as an end condition for the QPA* iterations. When $h(t)=t$, if $t \neq d_i$, then the iterations can be stopped. But with this condition, we have to judge whether t is an absolute deadline every time when $h(t)=t$, if t is not an absolute deadline, then we still have to find $\max\{d_i \mid d_i < t\}$ and let $t \leftarrow \max\{d_i \mid d_i < t\}$, hence the complexity is increased.

Lemma 6 For a schedulable system, if there exists any point with $h(t_\xi)=t_\xi$ in any checking interval, then during the iterations of QPA or QPA*, t takes the value of t_ξ.

Proof. Let $(0,L_x)$ be any time interval, define $d_\lambda = \max\{t \mid 0<t<L_x \wedge h(t)=t\}$, then at the beginning of the QPA* iterations, $t \geq d_\lambda$; During the QPA* iterations, when $t \geq d_\lambda$, we have $h(t) > d(t_\lambda) = d_\lambda$, hence the value of t is always greater or equal to d_λ, and t will take the value of d_λ. From Lemma 5, for a schedulable task set, any $t \in \{t \mid h(t)=t\}$ is an absolute deadline. So after t takes the value of d_λ, t moves backward to the previous deadline, define $d_\beta = \max\{t \mid 0<t<d_\lambda \wedge h(t)=t\}$, again t will take the value of d_β, and no time t with $h(t)=t$ in the interval $(0,L_x)$ will be missing. □

Theorem 8 For a schedulable system, at most one more $h(t)$ calculation is required by QPA* compared with QPA.

Proof. When $t \in [L_x, L)$, the required $h(t)$ calculations by QPA* and QPA are the same. So we only need to focus on the interval when $t \in (0,L_x)$. First we only consider the situation that $h(t)$ is always less than t during the iterative process.

Define $t_1 = \max\{d_i \mid d_i < L_x\}$, $t_2 = h(t_1)$, $t_3 = h(t_2)$,…, $t_m = h(t_{m-1})$, suppose $t_{m-1} > d_{\min}$ and $t_n \leq d_{\min}$, therefore QPA* needs $m-1$ jumps ($h(t)$ calculations) when $t \in (0,L_x)$. For the QPA test, suppose t_1' be the last value of t when $t \geq L_x$, then we have $t_1' \geq L_x > t_1$. Let $t_2' = h(t_1')$, $t_3' = h(t_2')$, …, and $t_m' = h(t_{m-1}')$. Since $h(t)$ is non-decreasing with t, we have $t_2' \geq t_2$, $t_3' \geq t_3$, …, $t_{n-1}' \geq t_{n-1} > d_{\min}$. Therefore QPA needs at least $m-2$ jumps when $t \in (0,L_x)$.

Then we consider the situation when there exists one or more points $h(t)=t$ during the interval $(0,L_x)$. Define $t_k = \max\{t \mid h(t) = t \wedge 0 < t < L_x\}$, from Lemma 6, t takes the value of t_k for both of QPA* and QPA. Let $t_1 > t_2 ... > t_{k-1} > t_k$, and $t_k = h(t_{k-1})$, then QPA* needs $k-1$ jumps before $t = t_k$. We can apply the same discussion to prove that during the iterative process of QPA, $t_2' \geq t_2$,…, $t_{k-1}' \geq t_{k-1}$, $t_k' \geq t_k$, so QPA needs at least $k-2$ jumps before $t=t_k$ in the interval $(0,L_x)$. After $t=t_k$, the required calculations by QPA* and QPA are the same.

From the above discussions, at most one more $h(t)$ calculation is required by QPA* compared with QPA for a schedulable system. □

Property 1 Let $(0,L_\zeta)$ be any checking interval, and $d_m = \max\{d_i \mid d_i < L_\zeta\}$. For a unschedulable system, if $h(d_m) \leq d_m$, then during the iterations of QPA or QPA*, before a missed deadline is found, $h(t)=t$.

Proof. Let $d^\Lambda = \max\{d_i \mid 0 < d_i < L_\zeta \wedge h(d_i) > d_i\}$, and $d' = \min\{d_i \mid d_i > d^\Lambda\}$. There are two cases.

Case 1. If $h(d') \neq d'$; from Lemma 4, $\forall t \in [h(d^\Lambda), d_m)$, we have $h(d^\Lambda) \leq h(t) \leq t$. Since d' is the only possible point (when $h(d')=d'$) in $(h(d^\Lambda), d_m)$ to let the value of t jump across the point $h(d^\Lambda)$ to d^Λ, in this case, t is always greater than or equal to $h(d^\Lambda)$, and t will take the value of $h(d^\Lambda)$. So $h(t)=t$ before t takes the value of d^Λ.

Case 2. If $h(d')=d'$; from Lemma 6, t takes the value of d'; therefore $h(t)=t$ before the missed deadline is found. □

6.1 More Than One L_x for QPA*.

Before addressing the issue of obtaining an effective value for L_x, we observe that the interval $(0,L)$ can be divided into more than two intervals, and the schedulability checked in each interval. If there exists a failure point in $(0,L)$, it must be found by QPA* in a test interval.

Define $\{L_{x1}, L_{x2}, ..., L_{xk}\}$ to be the *dividing points* for QPA*, and $\forall L_{xi} \in \{L_{x1}, L_{x2}, ...\}$ to be a dividing point, then a system is schedulable if and only if QPA* cannot find any failure point in these intervals. When there are k dividing points for QPA*, the intervals are checked by $(0,L_{x1})$, (L_{x1},L_{x2}),… (L_{xk},L), and the algorithm of Theorem 7 is changed to:

```
*find* $\leftarrow$ false;  $t \leftarrow \max\{d_i \mid d_i < L_{x1}\}$;
***while*** ( $h(t) > d_{\min} \wedge find =$ false)
  { ***if*** ( $h(t) < t$ )  $t \leftarrow h(t)$;
    ***else if*** ( $h(t) > t$ )  *find* $\leftarrow$ true;
    else  $t \leftarrow \max\{d_i \mid d_i < t\}$;
    }
if ( $find$ = false)
  { $t \leftarrow \max\{d_i \mid d_i < L_{x2}\}$;
    while ( $h(t) > L_{x1} \wedge find$ = false)
       { if ( $h(t) < t$ )  $t \leftarrow h(t)$;
         else if ( $h(t) > t$ )  $find \leftarrow$ true;
       else  $t \leftarrow \max\{d_i \mid d_i < t\}$;
       }}
else goto end;
……
if ( $find$ = false)
  { $t \leftarrow \max\{d_i \mid d_i < L\}$;
    while ( $h(t) > L_{xk} \wedge find$ = false)
       { if ( $h(t) < t$ )  $t \leftarrow h(t)$;
         else if ( $h(t) > t$ )  $find \leftarrow$ true;
       else  $t \leftarrow \max\{d_i \mid d_i < t\}$;
       }}
end:
if ( $find$ = false)   the system is schedulable;
else   the system is unschedulable;
```

The above algorithm can be proved by the same process of Theorem 7's proof.

Theorem 9 For a schedulable system, when there are k (k is a positive integer number) dividing points for QPA*, if QPA needs m times $h(t)$ calculations to complete a test, then QPA* needs at most $m+k$ times.

Proof. From Theorem 8's proof, each dividing point requires at most one more $h(t)$ calculation, hence at most k more $h(t)$ calculations are required when there are k dividing points. □

7. Experimental Evaluations

This section evaluates the performance of QPA* by intensive experiments on a large number of randomly generated task sets. Before the evaluations, the values and the number of dividing points is an issue for the efficiency of QPA*. We provide an empirical approach [15] based on the distribution of the largest failure point to find a suitable value of L_x for QPA*. From the experimental investigation of [15], the best value of L_x is changed according to variable conditions, and hence there is no single optimal value of L_x. In order to get the best performance of QPA*, we may need to change the values of dividing points and the number of dividing points according to the task sets under study.

However most of the time we do not know the properties of a task set before we test it. The process to decide the best value of L_x for each task set involves an extensive amount of simulation and is unnecessary due to the high efficiency of QPA. What we need to do is to choose a fixed value of L_x and a fixed number of dividing points for all kinds of task sets.

According to the empirical investigation [15] and Theorem 9, we conclude that 2 dividing points is better for QPA*, with $L_{x1} = 0.12L$ and $L_{x2} = 0.36L$. Most ($>60\%$) unschedulable task sets can find d_x in those two intervals, and from Theorem 9, at most two extra $h(t)$ calculations are needed for a schedulable system.

We compare the number of calculations required by QPA* with one or two dividing points. From the experiments of [15], when there is one dividing point for QPA*, we set a number of typical values for L_x, and when there are two dividing points for QPA*, we set $L_{x1} = 0.12L$ and $L_{x2} = 0.36L$. Since more dividing points can increase the calculations for schedulable systems, we do not compare the performance of QPA* with more than two dividing points. As the values of the dividing points are fixed for all kinds of task sets, there is no additional calculation needed by QPA* compared with QPA. Hence a reasonable metric to compare the results is to measure the number of times $h(t)$ has to be calculated.

In the experiments, the task sets are automatically generated by the following policies [16].

Utilizations generation policy.

In order to get a uniform distributed task utilizations in the range 0-1, we use the UUniFast algorithm [4] to generate the task utilizations. Bini and Buttazzo [4] showed that the UUniFast algorithm can efficiently generate task utilizations with uniform distributions.

Periods generation policy.

If we want to explore say 6 orders of magnitude (e.g. 1 - 1,000,000), then a set of random choices within this range will result in 99% of values being in the range 10,000-1,000,000 (only 3 orders of magnitude are actually explored in the expected 6 orders). Hence the task periods are generated according to an exponent distribution which is similar to the policy used in [11].

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 randomly generated in each interval, and one of the rest ($(n-1) \bmod k$) task periods is generated randomly from each intervals.

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 $b = 1.2 \times T_i$.

7.1 Evaluations for Unschedulable Task Sets

As QPA* is aimed at reducing the required calculations for unschedulable task sets, in this section, we compare the performance of QPA* for unschedulable task sets only.

In Figure 4-6, we compare the original QPA scheme with QPA* when different values of dividing points are used. Each point on the result diagrams are the average of 8,000 randomly generated unschedulable task sets.

Impact of the tasks number.

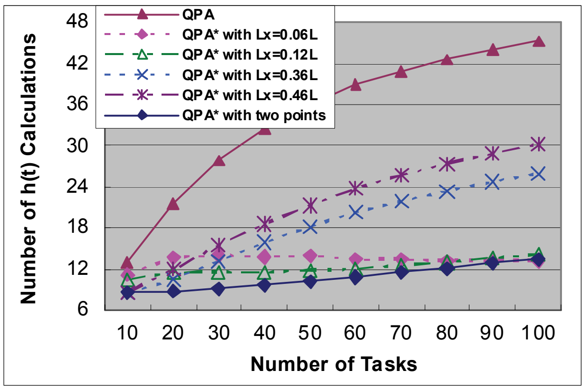

Figure 4. Impact of the tasks number for unschedulable task sets when U=0.96 and Tmax/Tmin=100.

Impact of the task periods range.

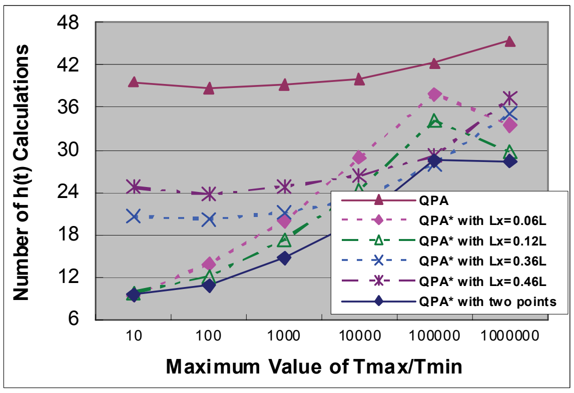

Figure 5. Impact of the task periods range for unschedulable task sets when U=0.96, n=60.

Impact of the task set's utilization.

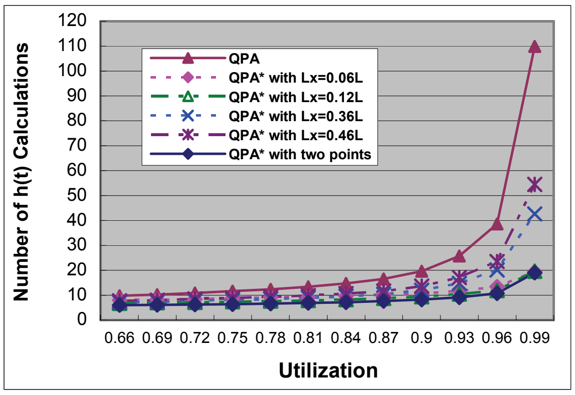

Figure 6. Impact of the processor utilization for unschedulable task sets when n=60, Tmax/Tmin=100.

From Figure 4-6, we can see that QPA* with two dividing points is nearly always the best and in general reduces the required calculations for schedulability.

7.2 Evaluations for General Task Sets

This section compares the performance of QPA* for general systems which contain schedulable and unschedulable task sets. All the task sets are automatically generated according to the default policies introduced at the beginning of this section. Each point on the result diagrams are the average of 8,000 randomly generated task sets.

Impact of the number of tasks.

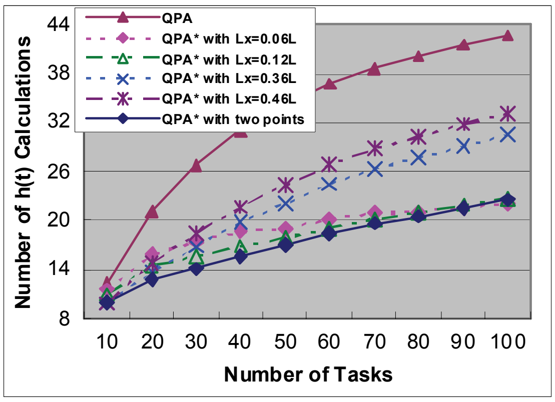

Figure 7. Impact of the number of tasks for general task sets when U=0.96, Tmax/Tmin=100.

The percentage of the schedulable task sets corresponds to Figure 7 is shown as the following graph.

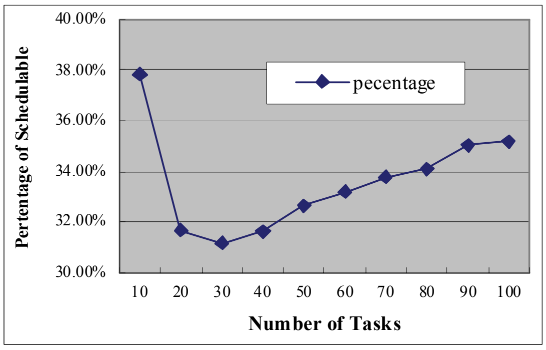

Figure 8. Percentage of schedulable task sets when U=0.96, Tmax/Tmin=1000.

Impact of the task periods range.

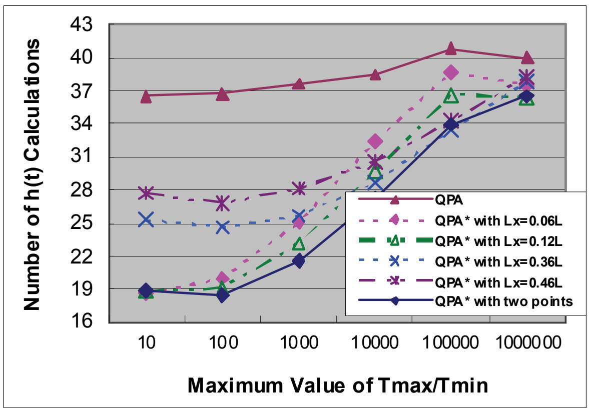

Figure 9. Impact of the task periods range for general task sets when U=0.96, n=60.

The percentage of the schedulable task sets corresponds to Figure 9 is shown as the follow graph.

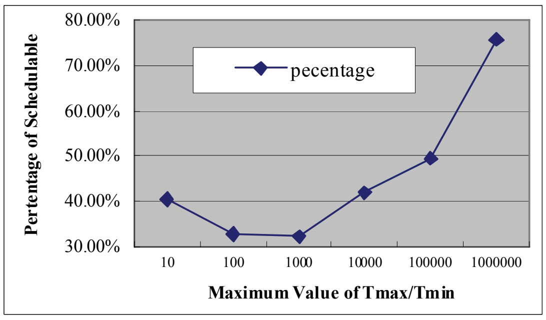

Figure 10. Percentage of the schedulable task sets when U=0.96, n=60.

Impact of the task set's utilization.

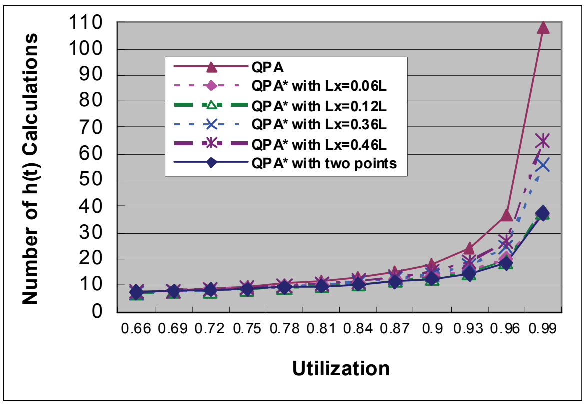

Figure 11. Impact of the processor utilization for general task sets when n=60, Tmax/Tmin=100.

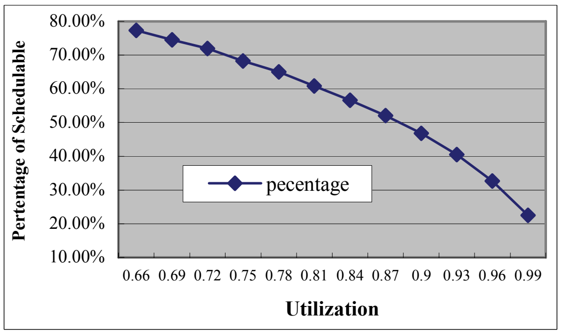

Figure 12. Percentage of the schedulable task sets when n=60, Tmax/Tmin=100.

More experimental results can be found in [15].

7.3 Comparing Other Values for QPA*

In order to observe how sensitive are the results for QPA* with two dividing points, we compare three sets of values. One set with $L_{x1} = 0.12L$, $L_{x2} = 0.36L$, one with $L_{x1} = 0.10L$ and $L_{x2} = 0.40L$, and one set with $L_{x1} = 0.08L$ and $L_{x2} = 0.30L$.

From the experiments of [15], we observed that the results are not sensitive to the dividing points' values when two points are employed for QPA*. The differences between the experimental results on each level of task parameters are less than one $h(t)$ calculation in nearly all situations.

7.4 Conclusion from the Experiments

QPA* can significantly decreases the required calculations to perform an exact schedulability test for general systems. Based on a large number of randomly generated task sets, QPA* had 1/3 the computation effort compared with QPA and at best reduced to 1/5 the effort required. In more than 95% of all experiments QPA* was better than QPA. Also QPA* with two dividing points had better performance than QPA* with one dividing point in almost all situations. The performance of QPA* is not sensitive to the values of the dividing points when two dividing points are set, $L_{x1} = 0.12L$ and $L_{x2} = 0.36L$ are suitable values for QPA*.

Although this paper has been concerned with improvement to QPA, it is important not to forget that for a typical task set that required, say 12 $h(t)$ calculations, the best non-QPA scheme is required to check all deadlines needing some 858,000 $h(t)$ calculations [14].

8. Conclusion

In this paper, we propose Improved Quick Processor-demand Analysis (QPA*) which builds on QPA. By intensive experiments, we show that QPA* can significantly decrease the required calculations for the schedulability test of unschedulable systems. A typical improvement of QPA* is 1/3 of the previous effort. We prove that the computation and load for schedulable systems remains nearly the same as QPA. Hence the required calculations for general systems can be significantly decreased.

According to experiments of Section 7, Theorem 9, and the empirical investigations [15], we suggest that two dividing points are used in QPA*, the values are $L_1 = 0.12L$ and $L_2 = 0.36L$. From the experimental results of Section 7, those two values significantly improve the performance of the schedulability test for general systems.

Appendix: Optimize the implementation of QPA*

In the QPA and QPA* algorithms, when $h(t) = t$, we need to find $\max\{d_i \mid d_i < t\}$ and give its value to t, to let the iterative process continue. The calculation of $\max\{d_i \mid d_i < t\}$ has the complex $O(n)$, this is not strictly necessary for the QPA or QPA* test when implementing QPA using integer arithmetic. We only need to let $t \leftarrow t-1$ to substitute $t \leftarrow \max\{d_i \mid d_i < t\}$, then we can get the same result for schedulability.

Theorem 10 In the QPA or QPA* algorithm, $t \leftarrow \max\{d_i \mid d_i < t\}$ can be substituted by $t \leftarrow t-1$; and $t \leftarrow \max\{d_i \mid d_i < L\}$ can be substituted by $t \leftarrow L-1$.

Proof. When $h(t) = t$, if t is an absolute deadline, as $h(t-1) = h(d_i^m)$, where $d_i^m = \max\{d_i \mid d_i < t\}$, then $t \leftarrow \max\{d_i \mid d_i < t\}$; if t is not an absolute deadline, then $h(t-1) = h(t) > t-1$, from Lemma 5, the system is unschedulable, so $t \leftarrow t-1$ has the same result as the original algorithm.

From Theorem 3 and Theorem 5, a missed deadline must be found in $(0, L)$ for a unschedulable system. Let $d_m = \max\{d_i \mid d_i < L\}$, since $h(L-1) = h(d_m)$, if $h(d_m) \le d_m$, then $t \leftarrow h(d_m)$; if $h(d_m) > d_m$ and $h(L-1) \le L-1$, then $h(t) = t = h(d_m)$, and $t \leftarrow t-1$, so $h(t-1) > t-1$; if $h(L-1) > L-1$, then the iteration stops. □

Theorem 11 For a schedulable system, QPA* with the optimization of Theorem 10 is always better than the original algorithm.

Proof. Let $d_i^m = \max\{d_i \mid d_i < t\}$. For a schedulable system, since $h(t-1) = h(d_i^m) \le d_i^m$, then $t \leftarrow h(d_i^m) \le d_i^m$ for both the algorithms. □

The calculation for $\max\{d_i \mid d_i < t\}$ has the same complex with $h(t)$, if we regard the computation for $\max\{d_i \mid d_i < t\}$ as one $h(t)$ calculation, then we can prove that for the $h(t-1)$ algorithm, at most one more $h(t)$ calculation is required for unschedulable system; this situation could only happen when a missed deadline is found immediately before a deadline satisfying $h(d_i) = d_i$.

Theorem 12 For a unschedulable system, when only $h(d_i) = d_i$ followed by a missed deadline, the QPA* algorithm with Theorem 10 require one more $h(t)$ calculation.

Proof. When $h(t) = t$ is not an absolute deadline, then $h(t-1) = h(t) > t-1$, so the required calculation is less than the original QPA* algorithm.

When $h(t) = t$ is an absolute deadline, and $\max\{d_i \mid d_i < t\}$ is not a missed deadline, then $t \leftarrow h(t-1) = h(d_i^m) \le d_i^m$, where $d_i^m = \max\{d_i \mid d_i < t\}$, hence this is equivalent to the original algorithm.

When $h(t) = t$ is a deadline, and d_i^m is a missed deadline, then $t \leftarrow h(t-1) = h(d_i^m)$, and $h(t) = t = h(d_i^m)$, then we have $t-1 < h(t-1)$, so 3 $h(t)$ calculations required compared only 2 with the previous $t \leftarrow \max\{d_i \mid d_i < t\}$. □

Since the situation described in Theorem 12 happens rarely, the optimized QPA* with Theorem 10 can decrease the required calculations for the vast majority of unschedulable systems and all schedulable systems, and at most one more $h(t)$ calculation is required by a unschedulable system in a particular case.

9. Acknowledgements

This work was funded in part by the EU FRESCOR and ArtistDesign projects. The authors would like to thank Sanjoy Baruah for his helpful discussion on an initial phase of this work.

10. References

[1] 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.

[2] 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.

[3] S.K. Baruah, R.R. Howell, and L.E. Rosier. Feasibility Problems for Recurring Tasks on One Processor. *Theoretical Computer Science*, 118(1):3-20, 1993.

[4] E. Bini and G.C. Buttazzo. Measuring the Performance of Schedulability Tests. *Journal of Real-Time Systems*, 30(1-2):129-154, 2005.

[5] G.C. Buttazzo. *Real-Time Scheduling and Resource Management*, *In Handbook of Real-Time and Embedded Systems*. 2008, Chapman & Hall/CRC.

[6] M.L. Dertouzos. Control Robotics: The Procedural Control of Physical Processes. *In Proceedings of the IFIP Congress*, pp.807-813, 1974.

[7] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive Real-Time uniprocessor Scheduling. *Technical Report 2966, INRIA, France*, 1996.

[8] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson. Computing the Minimum EDF Feasible Deadline in Periodic Systems. *In Proceedings 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.125-134, 2006.

[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 the ACM*, 20(1):40-61, 1973.

[11] R.I.Davis, A. Zabos, and A.Burns. Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems. *IEEE Transactions on Computers*, 57(9):1261-1276, 2008.

[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] M. Spuri. Analysis of Deadline Schedule Real-time Systems. *Technical Report 2772, INRIA, France*, 1996.

[14] F. Zhang and A. Burns. Schedulability Analysis for Real-Time Systems with EDF Scheduling. *Technical Report YCS-426-2008, Dept. of Computer Science, University of York, UK*, 2008.

[15] F. Zhang and A. Burns. Improvement to Quick Processor-demand Analysis for EDF-Scheduled Real-Time Systems. *Technical Report YCS-433-2008, Dept. of Computer Science, University of York, UK*, 2008.

[16] F. Zhang and A. Burns. Schedulability Analysis for Real-Time Systems with EDF Scheduling. *IEEE Transactions on Computers*, 2009.