General and Efficient Response Time Analysis for EDF Scheduling

Nan Guan and Wang Yi

Northeastern University, China
Uppsala University, Sweden

***Abstract*—Response Time Analysis (RTA) is one of the key problems in real-time system design. This paper proposes new RTA methods for EDF scheduling, with general system models where workload and resource availability are represented by request/demand bound functions and supply bound functions. The main idea is to derive response time upper bounds by lower-bounding the slack times. We first present a simple over-approximate RTA method, which lower bounds the slack time by measuring the "horizontal distance" between the demand bound function and the supply bound function. Then we present an exact RTA method based on the above idea but eliminating the pessimism in the first analysis. This new exact RTA method, not only allows to precisely analyze more general system models than existing EDF RTA techniques, but also significantly improves analysis efficiency. Experiments are conducted to show efficiency improvement of our new RTA technique, and tradeoffs between the analysis precision and efficiency of the two methods in this paper are discussed.**

I. Introduction

Response Time Analysis (RTA) is one of the most important problems in real-time system design. RTA is not only useful to perform local schedulability test on single-processors, but also plays important roles in the analysis of more complex real-time systems, e.g., distributed systems where the completion of a task generates outputs triggering computation or communication tasks on subsequent infrastructures [15], [6], [11], [12]. Since the completion time of the preceding task decides the release times of subsequent tasks, one can use RTA to bound the completion time of each task and decide the "release jitter" of the subsequent tasks.

EDF is a widely used real-time scheduling algorithms. Spuri [14] developed EDF RTA techniques for periodic tasks with extensions of jitters and sporadic bursts. It turns out that the RTA problem of EDF is more difficult than that of fixed-priority scheduling. Although RTA for fixed-priority scheduling has been extended to general workload and resource models represented by request bound functions (arrival curves) and supply bound functions (service curves) [10], no such work has been done for EDF to our best knowledge.

Spuri's EDF RTA technique relies on enumerating a (typically very large) number of concrete release patterns as the candidates of the worst-case scenario. On one hand, this requires high computational effort. On the other hand, this complicates its extension to more expressive models, as the technique needs to be customized to include new features of each new model. This is particularly hindering when workload and resource is specified in a rather abstract way (like the one considered in this paper).

978-3-9815370-2-4/DATE14/©2014 EDAA

In this paper we present EDF RTA methods for general models with workload and resource represented by request/demand bound functions and supply bound functions. These general models are used in design and analysis frameworks such as Real-Time Calculus [5] and SymTA/S [12]. The key insight is that in EDF it is easier to derive response time upper bounds indirectly by lower-bounding the slack times. This not only allows us to perform RTA with general workload and resource models, but also can greatly improve the analysis efficiency.

More specifically, we first present a simple over-approximate RTA method, which lower bounds the slack times by measuring the "horizontal distance" between the demand bound function and the supply bound function. Then we present an *exact* RTA based on the similar idea but eliminating the pessimism in the first analysis. Experiments show that the new RTA technique can greatly improve the analysis efficiency comparing with Spuri's RTA. We also discuss the tradeoff between the analysis precision and efficiency of the two methods proposed in this paper.

II. Preliminaries

A. Resource Model

We consider a processing platform with capacity characterized by a *supply bound function* $\mathsf{sbf}(\delta)$ [4], [13], which quantifies the minimal cumulative computation capacity provided by the processing platform within a time interval of length δ. The supply bound function is essentially the same as the lower service curve in Real-Time Calculus [5].

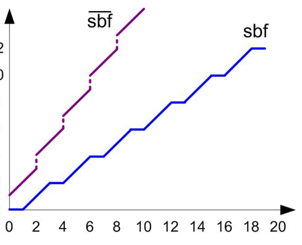
sbf is a continuous function. The pseudo-inverse function of sbf, denoted by $\overline{\mathsf{sbf}}$, characterizes the minimal interval length to provide a certain amount of computation capacity:

$$\overline{\mathsf{sbf}}(x) = \min\{\delta | \mathsf{sbf}(\delta) = x\}$$

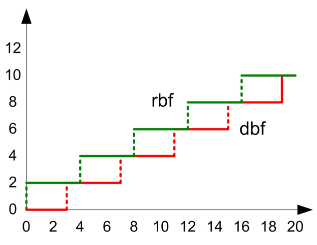
Figure 1-(a) illustrates sbf and its pseudo-inverse $\overline{\mathsf{sbf}}$ of a TDMA resource with a period of 4 and slot size of 3.

B. Workload Model

The task system τ consists of $\mathcal{N}$ independent tasks $\{\tau_1, \tau_2, \cdots, \tau_{\mathcal{N}}\}$. Each task τ_i releases infinitely many *jobs*.



(a) sbf and $\overline{\mathsf{sbf}}$



(b) rbf and dbf

Fig. 1. Illustration of sbf, $\overline{\mathsf{sbf}}$, rbf and dbf.

We use J_i^j to denote the j^{th} job released by task τ_i. For simplicity, we also omit the superscript and use J_i to denote a job released by τ_i when the context is clear.

Each task τ_i has a relative deadline D_i, which specifies the requirement that the computation demand of each job must be finished no later than D_i time units after its release time.

The workload of a task τ_i is characterized by a *request bound function* $\mathsf{rbf}_i(\delta)$ [1], which quantifies the maximum cumulative execution requests that could be generated by jobs of τ_i released within a time interval of length δ.

rbf_i is a staircase function, and it is essentially the same as the upper arrival curve in Real-Time Calculus [5] and SymTA/S [12]. As a common assumption in EDF scheduling analysis [2], [14], there exists a bounded number $\mathcal{L}'$ such that

$$\mathsf{rbf}(\mathcal{L}') \leq \mathsf{sbf}(\mathcal{L}')$$

which guarantees that in long term the system source supply is no smaller than the total execution demand. We let $\mathcal{L} = \mathcal{L}' + \mathcal{D}$ where $\mathcal{D}$ is the largest relative deadline among all tasks.

In the analysis of EDF scheduling, an important concept is the *demand bound function* dbf_i [2], which can be obtained by horizontally "shifting" rbf_i rightwards for D_i time units:

$$\mathsf{dbf}_i(\delta) = \begin{cases} 0 & \delta < D_i \\ \mathsf{rbf}_i(\delta - D_i) & \delta \geq D_i \end{cases} \quad (1)$$

Figure 1-(b) illustrates the rbf and dbf of a sporadic task [2] with period of 4, relative deadline of 3 and worst-case execution time of 2. We define $\mathsf{sbf}(\delta)$ and $\mathsf{dbf}(\delta)$ as the total demand bound function of the system:

$$\mathsf{rbf}(\delta) = \sum_{\forall \tau_i \in \tau} \mathsf{rbf}_i(\delta) \quad \text{and} \quad \mathsf{dbf}(\delta) = \sum_{\forall \tau_i \in \tau} \mathsf{dbf}_i(\delta)$$

Finally, we assume the number of points where rbf and dbf "steps" in a unit-length interval is bounded by a constant, to exclude the case that rbf/dbf curves are "infinitely complex".

C. EDF Scheduling and Worst-case Response Time

When a job of task τ_i is released at time t, its *absolute deadline* is at $t + D_i$. The task system is scheduled by EDF algorithm, which assigns priorities to *active* jobs (jobs that have been released but not finished yet) according to their absolute deadlines: the earlier deadline and higher priority. In case of multiple active jobs having the same absolute deadline, the EDF scheduler may prioritize *any* of them for execution. The aim of this paper is to calculate the worst-case response time R_i of each task:

Definition II.1 (Worst-Case Response Time)**.** *The worst-case response time R_i of a task τ_i is the length of the* longest *interval from a job's release till its completion.*

Note that we allow the case of $R_i > D_i$. A job finishes after its absolute deadline is called a *tardy* job [3]. Although a tardy job cannot finish computation before the expected deadline, it is still interesting to know its *tardiness*, i.e., how much it lags behind, in many soft real-time systems [3], [8].

D. Review of Spuri's RTA for EDF

Spuri [14] introduced a RTA method for sporadic tasks with jitters and sporadic bursts on a fully dedicated processor ($\mathsf{sbf}(\delta) = \delta$). For simplicity of presentation, we review Spuri's RTA technique with periodic tasks. Details about handling jitters and sporadic bursts can be found in [14].

Each task τ_i is characterized by three parameters: worst-case execution time C_i, relative deadline D_i and period T_i. The worst-case response time R_i of τ_i is calculated by:

$$R_i = \max_{\forall p \in P, a \in A_p,} \mathcal{R}_i(a, p)$$

where

$$P = [1, \lceil \mathcal{L}/T_i \rceil]$$

$$A_p = \{x - (p-1)T_i - D_i | x \in X \cap [(p-1)T_i + D_i, pT_i + D_i]\}$$

$$X = \bigcup_{\forall \tau_j \in \tau, q \in [1, \lceil \frac{\mathcal{L}}{T_j} \rceil]} \{(q-1)T_j + D_j\}$$

$$\mathcal{R}_i(a, p) = w(a, p) - a - (p-1)T_i$$

and $w(a, p)$ is the minimal solution of equation

$$w(a, p) = pC_i + \sum_{\tau_j \neq \tau_i} W_j(w(a, p),\ a + (p-1)T_i + D_i)$$

where $W_j(x, y) = \min(\lceil \frac{x}{T_j} \rceil, \lfloor \frac{y - D_j}{T_j} \rfloor + 1) \cdot C_j$.

Spuri's analysis is rather complicated, even with the simple sporadic task and fully dedicated resource model. Extending it to more expressive models could be difficult and error-prone. On the other hand, Spuri's analysis contains tremendous redundant computation, which leads to low analysis efficiency. The overall complexity of Spuri's RTA to a periodic task set is $O(\mathcal{N}\mathcal{T}\mathcal{L}'^2)$, where $\mathcal{N}$ is the total number of tasks, $\mathcal{T}$ is the maximal period among all tasks and $\mathcal{L}'$ is the maximal busy period size. The target of this paper is to overcome the above problems, by providing EDF RTA techniques that are more general, more efficient and easier to understand and remember.

III. Over-Approximate RTA

In this section, we first introduce a simple *over-approximate* RTA method. After presenting the analysis, we also use an example to explain why it may over-estimate the response time. Then in Section IV, we will reuse these insights and present our second RTA method which yields *exact* results.

Unlike traditional RTA techniques [7], [9], [14], our RTA method bounds the the response times *indirectly*: it calculates a *lower* bound on the *worst-case slack time* of the task, by which the response time upper bound can be easily obtained.

Definition III.1 (Worst-Case Slack Time)**.** *The worst-case slack time S_i of a task τ_i is the length of the* shortest *interval from a job's completion till its absolute deadline.*

The task's worst-case response time is calculated by:

$$R_i = D_i - S_i$$

Note that R_i is greater than D_i when S_i is negative.

We can safely lower-bound the slack time of each task by:

Theorem III.2. *The slack time of task τ_i is bounded from below by*

$$S_i^* = \min_{\forall \delta : \mathcal{L} \geq \delta \geq D_i} \left\{ \delta - \overline{\mathsf{sbf}}(\mathsf{dbf}(\delta)) \right\} \tag{2}$$

Proof: We prove the theorem by contradiction. Suppose at runtime task τ_i has a job J_i whose slack time, denoted by S', is strictly smaller than S_i^*. Let t_r, t_d and t_f be the release time, absolute deadline and finish time of J_i respectively. Let t_o be the earliest time instance before t_f such that at any time instant in $[t_o, t_f]$ there is at least one active job with deadline no later than t_d. Let $\ell = t_d - t_o$.

The total amount of workload (of jobs with deadline no later than t_d) in $[t_o, t_d]$ is bounded by $\mathsf{dbf}(\ell)$, and it takes the resource for at most $\overline{\mathsf{sbf}}(\mathsf{dbf}(\ell))$ time units to provide enough capacity to finish it. So we know there exist a time point in $[t_o, t_o + \overline{\mathsf{sbf}}(\mathsf{dbf}(\ell))]$ at which the processor is idle or executing jobs with deadline later than t_d. By the definition of t_o we know that this time point is not in $[t_o, t_f)$, so

$$t_o + \overline{\mathsf{sbf}}(\mathsf{dbf}(\ell)) \geq t_f \tag{3}$$

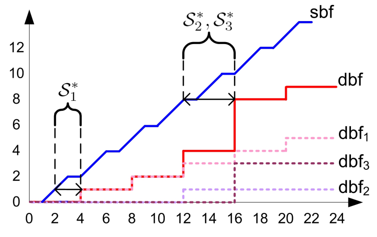
Since J_i itself is an active job at t_r, by the definition of t_o we know $t_o \leq t_r$ and thus $\ell \geq D_i$. On the other hand, the length of $(t_o, t_r]$ is bounded by $\mathcal{L}'$ and the length of $(t_r, t_d]$ is bounded by $\mathcal{D}$, so $\ell \leq \mathcal{L}$. In summary, $\mathcal{L} \geq \ell \geq D_i$. Then by (2) we have $S_i^* \leq \ell - \overline{\mathsf{sbf}}(\mathsf{dbf}(\ell)$, and by $S' < S_i^*$ we have $S' < \ell - \overline{\mathsf{sbf}}(\mathsf{dbf}(\ell)$, i.e., $t_d - S' > t_d - \ell + \overline{\mathsf{sbf}}(\mathsf{dbf}(\ell)$. Then we apply substitutes $t_o = t_d - \ell$ and $t_f = t_d - S'$ to get $t_f > t_o + \overline{\mathsf{sbf}}(\mathsf{dbf}(\ell)$, which contradicts (3). ■

Intuitively, the slack time lower bound stated in the above theorem is the minimal "horizontal distance" between dbf and sbf (in the range of $\delta \geq D_i$). Note that if $\mathsf{dbf}(\delta)$ is larger than $\mathsf{sbf}(\delta)$ for some $\delta > D_i$, then S_i^* is negative.

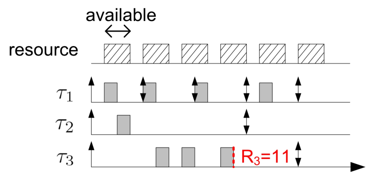
Example III.3. *Suppose task set τ consists of three sporadic tasks τ_1, τ_2 and τ_3 with parameters as follows: $\{C_1 = 1,\ T_1 = D_1 = 4\}$, $\{C_2 = 1,\ T_2 = D_2 = 12\}$ and $\{C_3 = 3,\ T_3 = D_3 = 16\}$. The task set is scheduled by EDF on a TDMA resource with period of 4 and slot size of 3. By Theorem III.2 we obtain the slack time lower bound of each tasks: $S_1^* = 2$ and $S_2^* = S_3^* = 4$, as illustrated in Figure 2-(a). So the worst-case response time of τ_1, τ_2 and τ_3 is bounded by $D_1 - 2 = 2$, $D_2 - 4 = 8$ and $D_3 - 4 = 12$, respectively.*

The slack time lower bound S_i^* is safe, but in general pessimistic, as shown in the following example.

Example III.4. *When we analyze τ_3 in the above example, the minimal horizontal distance between $\mathsf{sbf}(\delta)$ and $\mathsf{dbf}(\delta)$*



(a) The computation of S_i^*



(b) The pessimism of S_i^*

Fig. 2. Illustration of Example III.3 and III.4.

(only considering the part on $\mathsf{dbf}(\delta)$ with $\delta \geq D_3$) occurred at $\delta = 16$, with $\mathsf{dbf}(16) = 8$ and $\overline{\mathsf{sbf}}(\mathsf{dbf}(16)) = \overline{\mathsf{sbf}}(8) = 12$. The slack bound of τ_3 is $12 - 8 = 4$, and its response time bound is $D_3 - 4 = 12$. However, if we simulate the worst-case workload in a time interval of length 16 as in Figure 2-(b), the response time is actually 11 (which is indeed the worst-case response time of τ_3). This is because, although the workload of the last job of τ_1 is included in $\mathsf{dbf}(16)$, this job is actually released after the finish time of the analyzed job of τ_3 and thus does not really contribute to the interference.

Special Case where S_i^* is Exact

Although S_i^* is generally a pessimistic bound, it is indeed the exact answer in the following special case:

Theorem III.5. *S_i^* is the* exact *worst-case slack time of task τ_i if $S_i^* < 0$.*

Due to space limit we omit the formal proof but only provide the intuition: According to the standard dbf-based EDF schedulability test [2], a task τ_i is not schedulable iff sbf and rbf cross each other at some point no smaller than D_i, i.e., $S_i^* < 0$. When a job J_i finishes after its deadline, the jobs with release time and deadline in $[t_o, t_d]$ all can interfere with J_i. So the over-estimation problem in the above example does not exist, and $\mathsf{dbf}(t_d - t_o)$ precisely quantifies the total workload that needs to be finished before J_i is done.

When S_i^* is negative, τ_i is a *tardy* task. Theorem III.5 says that $|S_i^*|$ is the exact worst-case *tardiness* of a tardy task τ_i.

Algorithmic Implementation and Complexity

The computation of S_i^* for different tasks are essentially the same. The only difference is that for each task τ_i we only need to visit δ no smaller than its D_i. So we only need to "scan" the curve $\mathsf{dbf}(\delta)$ for one time, to compute the slack time bounds of *all* tasks in τ.

Algorithm 1 shows the pseudo-code of the algorithm to compute $S_1^*, \cdots, S_{\mathcal{N}}^*$. For simplicity of presentation, we assume all tasks have different relative deadlines, and tasks are sorted in increasing order of their relative deadlines. The case where multiple tasks have the same relative deadline can be easily handled with minor revisions of the presented algorithm. Intuitively, the algorithm first calculates the minimal "horizontal distance" between dbf and sbf with δ values in each segment $[D_i, D_{i+1})$, recorded by s_i. With these s_i, the slack time bound of all tasks can be calculated in $O(\mathcal{N})$ time.

```
1: $s_1 = s_2 = \cdots = s_{\mathcal{N}} = +\infty$
2: $i = 1$
3: **for** each candidate value of $\delta$ **do**
4:    **if** $\delta \geq D_{i+1}$ **then**
5:       $i \leftarrow i + 1$
6:    **end if**
7:    $s_i \leftarrow \min(s_i, \delta - \overline{\mathsf{sbf}}(\mathsf{dbf}(\delta)))$
8: **end for**
9: **for** $i \leftarrow \mathcal{N} \cdots 1$ **do**
10:    $S_i^* \leftarrow s_i$
11:    $s_{i-1} \leftarrow \min(s_i, s_{i-1})$ // does not execute when $i = 1$
12: **end for**
13: return $S_1^*, S_2^*, \cdots, S_{\mathcal{N}}^*$
```

Algorithm 1: Pseudo-code of the algorithm implementing Theorem III.2.

Since sbf is continuous and dbf is a staircase function, the candidate values of δ are the points in $[D_1, \mathcal{L}]$ at which $\mathsf{dbf}(\delta)$ is not differentiable i.e., where dbf "steps". So the number of candidate values of δ is bounded by $O(\mathcal{L})$ (recall that the number of points where dbf "steps" in a unit length is bounded by a constant). Moreover, the second for-loop iterates for $\mathcal{N}$ times. So the overall complexity of Algorithm 1 is $O(\mathcal{L}+\mathcal{N})$.

IV. Exact RTA

S_i^* is a pessimistic slack time lower bound since it ignores the fact that the workload released after the finish time of the analyzed job should not be included into the interference calculation. In this section we address this problem and present an exact RTA method.

As presented in last section, S_i^* is the exact worst-case slack time if τ_i is a tardy task (S_i^* is negative). So in this section we focus on the case that τ_i is *not* tardy, i.e., it holds

$$\forall \delta \geq D_i : \mathsf{dbf}(\delta) \leq \mathsf{sbf}(\delta) \tag{4}$$

We first define a task's *mixed bound function*:

Definition IV.1 (Mixed Bound Function). *For any $\delta \geq \gamma \geq 0$, the* mixed bound function *of τ_i is defined as*

$$\mathsf{mbf}_i(\delta, \gamma) = \min(\mathsf{dbf}_i(\delta), \mathsf{rbf}_i(\gamma))$$

Figure 3 illustrates $\mathsf{mbf}_i(\delta, \gamma)$. We consider two time intervals $[t_o, t_o + \delta]$ and $[t_o, t_o + \gamma]$ which both start at time t_o. $\mathsf{mbf}_i(\delta, \gamma)$ captures the workload of τ_i's jobs that are released in $[t_o, t_o + \gamma]$ and with deadline no later than $t_o + \delta$. In (a), while $\mathsf{dbf}(\delta)$ includes all of the five jobs, $\mathsf{sbf}(\gamma)$ only includes the first four jobs since the last job is released after $t_o + \gamma$, so $\mathsf{mbf}_i(\delta, \gamma)$ equals the total workload of the first four jobs. In

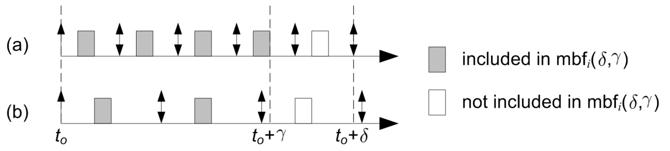

(a)
(b)
t_o
$t_o+\gamma$
$t_o+\delta$
included in $\mathsf{mbf}_i(\delta,\gamma)$
not included in $\mathsf{mbf}_i(\delta,\gamma)$

Fig. 3. Illustration of the mixed bound function $\mathsf{mbf}_i(\delta, \gamma)$.

(b), while all the three jobs are released in $[t_o, t_o + \gamma]$, $\mathsf{dbf}(\delta)$ excludes the last job as its deadline is later than $t_o + \delta$.

We define the total mixed bound function of the system:

$$\mathsf{mbf}(\delta, \gamma) = \sum_{\forall \tau_i \in \tau} \mathsf{mbf}_i(\delta, \gamma)$$

It should be noted that $\mathsf{mbf}(\delta, \gamma)$ does *not* necessarily equal $\min(\mathsf{dbf}(\delta), \mathsf{sbf}(\gamma))$. With this new function $\mathsf{mbf}(\delta, \gamma)$, we can compute a task's *exact* worst-case slack time:

Theorem IV.2. *The exact worst-case slack time of task τ_i is computed by*

$$S_i = \min_{\forall \delta: \mathcal{L} \geq \delta \geq D_i} \max_{\forall \gamma: \gamma \leq \delta} \{\delta - \gamma \,|\, \mathsf{mbf}(\delta, \gamma) \leq \mathsf{sbf}(\gamma)\} \tag{5}$$

Proof: First, we show S_i is well-defined in the sense that

$$\{(\delta, \gamma) | \mathcal{L} \geq \delta \geq D_i \wedge \gamma \leq \delta \wedge \mathsf{mbf}(\delta, \gamma) \leq \mathsf{sbf}(\gamma)\} \tag{6}$$

is *not* empty and thus (5) always returns an answer.

Let x be a number in $[D_i, \mathcal{L}]$, then by (4) we know

$$\mathsf{dbf}(x) \leq \mathsf{sbf}(x) \Rightarrow \mathsf{mbf}(x, x) \leq \mathsf{sbf}(x)$$

so any (δ, γ) s.t. $\mathcal{L} \geq \delta \geq D_i \wedge \delta = \gamma$ is contained in (6).

In the following we prove S_i is both a *safe* and *tight* lower bound of τ_i's slack time.

Safety: We shall prove that the slack time of any job released by τ_i is no smaller than S_i. We prove it by contradiction. Suppose J_i is a job of task τ_i with slack time $S' < S_i$. Let t_r, t_d and t_f be the release time, absolute deadline and finish time of J_i respectively. Let t_o be the earliest time instance before t_f such that at any time instant in $[t_o, t_f]$ there is at least one active job with deadline no later than t_d. Let $\delta' = t_d - t_o$. Since J_i is active at t_r, we know $t_o \leq t_r$ and thus $\delta' \geq D_i$. The length of $[t_o, t_r]$ is bounded by $\mathcal{L}'$ and the length of $[t_r, t_d]$ is bounded by $\mathcal{D}$, so $\delta' \leq \mathcal{L} = \mathcal{L}' + \mathcal{D}$. In summary, $\mathcal{L} \geq \delta' \geq D_i$. Then by the definition of S_i and $S' < S_i$ we have

$$S' < \max_{\gamma \leq \delta'} \{\delta' - \gamma \mid \mathsf{mbf}(\delta', \gamma) \leq \mathsf{sbf}(\gamma)\}$$

Let γ' be the *smallest* assignment of γ s.t. $\mathsf{mbf}(\delta', \gamma) \leq \mathsf{sbf}(\gamma)$ (thus $\delta' - \gamma$ is maximized), then we have

$$S' < \delta' - \gamma' \Rightarrow t_d - S' > t_d - \delta' + \gamma' \Rightarrow t_f > t_o + \gamma' \tag{7}$$

On the other hand, since $\mathsf{mbf}(\delta', \gamma') \leq \mathsf{sbf}(\gamma')$, the workload of all jobs released in $[t_o, t_o + \gamma']$ and with deadline no later than $t_o + \delta' = t_d$ have been finished by $t_o + \gamma'$, and particularly, J_i has been finished by $t_o + \gamma'$, which contradicts (7).

Tightness: We shall construct a scenario where the slack time of a job of τ_i is exactly S_i. Let δ' and γ' be the assignments of δ and γ that give the value of S_i in (5). Let each task release its first job at time t_o and release as much workload as possible since then (following its rbf_i). Moreover, we move the release time of the last job of the analyzed task τ_i released in $[t_o, t_o + \delta' - D_i]$, denoted by J_i, such that its deadline aligns with $t_o + \delta'$. Note that $[t_o, t_o + \delta' - D_i]$ is well-defined since $\delta' \geq D_i$. Let J_i have the lowest priority among all the jobs with deadlines at $t_o + \delta'$. Since γ' is the assignment of γ maximizing $\delta' - \gamma$ with this particular δ', we know:

$$\forall \gamma'' < \gamma' : \mathsf{mbf}(\delta', \gamma'') > \mathsf{sbf}(\gamma'') \quad (8)$$

Under the particular release pattern described above, $\mathsf{mbf}(\delta', \gamma'')$ is the exact total workload of jobs (of all tasks) released in $[t_o, t_o + \gamma'')$ and having priority no lower than J_i. By (8) we know $t_o + \gamma'$ is the first time point after t_o at which the processor is idle or executing jobs with priority lower than J_i. Therefore J_i is finished exactly at $t_o + \gamma'$ and its slack time equals $\delta' - \gamma'$, i.e., equals S_i. ■

By examining (5) we can get a general property of EDF scheduling concerning the relation of tasks' relative deadlines and worst-case slack times:

Corollary IV.3. *For any two tasks τ_i and τ_j in a task set τ scheduled by EDF, it holds*

$$D_i = D_j \Rightarrow S_i = S_j \quad \text{and} \quad D_i > D_j \Rightarrow S_i \geq S_j$$

Algorithmic Implementation and Complexity

```
1: $s_1 = s_2 = \cdots = s_{\mathcal{N}} = +\infty$
2: $i = 1$
3: for each candidate value of $\delta$ (in increasing order) do
4:    if $\delta \geq D_{i+1}$ then
5:       $i \leftarrow i + 1$
6:    end if
7:    if $\delta - \overline{\mathsf{sbf}}(\mathsf{dbf}(\delta)) < s_i$ then
8:       $\gamma_{old} \leftarrow 0$
9:       $\gamma_{new} \leftarrow \overline{\mathsf{sbf}}(\mathsf{mbf}(\delta, 0))$
10:      while $\gamma_{new} \neq \gamma_{old}$ do
11:         $\gamma_{old} \leftarrow \gamma_{new}$
12:         $\gamma_{new} \leftarrow \overline{\mathsf{sbf}}(\mathsf{mbf}(\delta, \gamma_{old}))$
13:      end while
14:      $s_i \leftarrow \min(s_i, \delta - \gamma_{new})$
15:   end if
16: end for
17: for $i = \mathcal{N} \cdots 1$ do
18:    $S_i \leftarrow s_i$
19:    $s_{i-1} \leftarrow \min(s_i, s_{i-1})$ // does not execute when $i = 1$
20: end for
21: return $S_1, S_2, \cdots, S_{\mathcal{N}}$
```

Algorithm 2: Pseudo-code of the algorithm implementing Theorem IV.2.

A naive way to compute S_i using (5) is enumerating all the combinations of δ and γ candidate values satisfying $\mathcal{L} \geq \delta \geq D_i$ and $\delta \geq \gamma$. This could be inefficient when $\mathcal{L}$ is large. Algorithm 2 presents the pseudo-code for a more efficient implementation.

Similar to Algorithm 1, Algorithm 2 also integrates the slack time bounds computation of *all* tasks, using s_i to keep track of the local minimum in each segment $[D_i, D_{i+1})$. Algorithm 2 and 1 differ in the first *for-loop*. Two optimizations are applied to speedup the analysis. Firstly, to compute

$$\max_{\forall \gamma : \gamma \leq \delta} \{\delta - \gamma | \mathsf{mbf}(\delta, \gamma) \leq \mathsf{sbf}(\gamma)\}$$

with a particular δ, we use the well-known iterative fixed-point calculation technique [7] instead of searching over all possible γ values (line 8 to 14).

Secondly, for each candidate value of δ, we first check whether $\delta - \overline{\mathsf{sbf}}(\mathsf{dbf}(\delta))$ is smaller than the current s_i. If not, then we can safely skip further calculation regarding this δ value. This is because $\delta - \overline{\mathsf{sbf}}(\mathsf{dbf}(\delta))$ is a lower bound of the slack time with this particular δ. If this lower bound is already greater than s_i, then using $\mathsf{mbf}(\delta, \gamma)$ to further refine this bound (make it potentially bigger) will not yield a result smaller than s_i. This optimizations is very effective in practise. Typically, the worst-case slack time of a task occurs with relatively small δ values. Since we check the δ candidates in increasing order, we can obtain small slack time estimations with smaller δ values, and skip the fixed-point computation regarding γ with large δ values.

Similar to Algorithm 1, the candidate values of δ are the points in $[D_1, \mathcal{L}]$ where dbf "steps". The overall complexity of Algorithm 2 is $O(\mathcal{L}^2 + \mathcal{N})$.

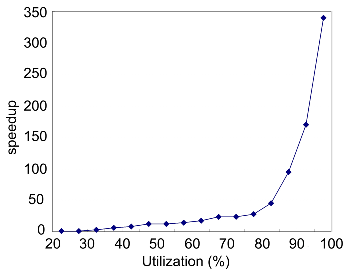


Fig. 4. Speedup of our exact RTA comparing with Spuri's method.

V. Evaluation

A. Analysis Efficiency Improvement over Spuri's RTA

We first evaluate the analysis efficiency improvement of the exact slack (response) time analysis in Section IV comparing with Spuri's method in [14]. We adopt the sporadic task model [2] and a fully dedicated processor ($\mathsf{sbf}(\delta) = \delta$), with which Spuri's method is also applicable. Tasks are randomly generated, with the periods (T_i) uniformly distributed in $[100, 1000]$, utilizations (C_i/T_i) uniformly distributed in $[0.01, 0.2]$, and the ratio between the relative deadline and period (D_i/T_i) uniformly distributed in $[0.8, 1]$. We generate task sets as follows: A task set of 2 tasks is generated and analyzed by both our exact RTA and Spuri's RTA. Then we increase the number of tasks by 1 to generate a new task set. This process is iterated until the total utilization exceeds 100%. The whole procedure is then repeated, starting with a new task set of 2 tasks, until a reasonably large sample space has been generated.

Figure 4 shows the *speedup* of our new exact RTA over Spuri's. For each task set, the *speedup* is the ratio between the analysis time by Spuri's RTA and that by our exact RTA. A point in the curve represents the average *speedup* of all

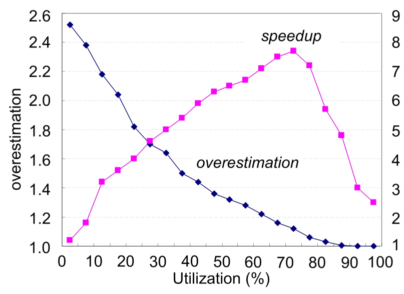


Fig. 5. Comparison of the two RTA methods of this paper.

task sets generated in a certain scope of system utilization corresponding to the abscissa.

From Figure 4 we can see that the analysis efficiency improvement of our new RTA over Spuri's RTA is significant, especially with task sets of high total utilizations. The reason is twofold. First, Spuri's RTA analyzes each task separately, while our RTA integrates the analysis of all tasks and only "scans" the demand bound function curve once. A task set with higher utilization typically contains more tasks, so the efficiency improvement of our RTA is greater. Second, when the task set's total utilization is close to 100%, $\mathcal{L}$ is typically very large. The efficiency of Spuri's RTA is sensitive to $\mathcal{L}$: the number of different combinations to be checked by Spuri's RTA grows rapidly as $\mathcal{L}$ increases. However, thanks to the optimization of skipping the refinement with δ values satisfying $\delta - \overline{\mathsf{sbf}}(\mathsf{dbf}(\delta)) < s_i$ (see the last second paragraph of Section IV), our exact RTA can skip the computation between line 8 and line 14 of Algorithm 2 for most large δ values, and thus is less sensitive to the growth of $\mathcal{L}$.

B. Comparison of the Two Methods in This Paper

In the following we compare the two RTA methods of this paper, regarding their efficiency and precision. We use the same strategy as above to generate task sets. In Figure 5, each point on curve *overestimation* represents the average ratio between the response time bounds obtained by the approximate RTA and by the exact RTA for task sets in a certain total utilization scope. Each point on curve *speedup* represents the average ratio between the analysis time of the exact RTA and the approximate RTA in a certain total utilization scope.

By curve *overestimation* we see that the precision of the approximate RTA increases as the total utilization increases. This is because in task sets with higher total utilization, tasks' response times are typically closer to relative deadlines, so it has lower chance for the approximate RTA to mis-include the workload released after the analyzed job's finish time.

By curve *speedup* we see that the efficiency gap between the exact and the approximate RTA methods is smaller for task sets with either very low or very high total utilization. When the task set has very low total utilization, the iterative fixed-point calculation regarding γ typically converges very quickly, so the analysis time by the two methods are close. When the task set has very high total utilization, $\mathcal{L}$ is typically very large. As we discussed in last subsection, the exact RTA skips the fixed-point calculation regarding γ for a large portion of δ values. In other words, for most δ values the computation effort of both methods are the same, so the gap between their overall analysis time is small.

From the above results we can see that as the total utilization increases, the approximate RTA becomes more precise and gets more rewards in efficiency. For tasks with very high total utilization ($\geq 80\%$), the approximate RTA is almost as precise as the exact RTA, but at the same time the efficiency of the exact RTA is also catching up. It's up to the system designer to choose the proper analysis method according to their efficiency and precision requirements. But at least one can draw a clear conclusion that for task systems with very low total utilization, it does not make much sense to use the approximate RTA as it leads to rather imprecise results but benefits little in efficiency.

VI. Conclusions

RTA is not only useful in local schedulability test, but also lends itself to many complex design and analysis problems. This paper proposes new RTA methods for EDF, with general system models represented by request/demand bound functions and supply bound functions. Our new RTA method, not only allows to precisely analyze more general system models than existing EDF RTA techniques, but also significantly improves analysis efficiency.

Acknowledgement

This work is partially supported by NSF of China (Grant No. of 61300022 and 61370076).

References

[1] S. Baruah. Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks. In *Real-Time Systems*, 2003.
[2] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *RTSS*, 1990.
[3] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *RTSS*, 2005.
[4] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *RTSS*, 2007.
[5] L. Thiele et. al. A framework for evaluating design tradeoffs in packet processing architectures. In *DAC*, 2002.
[6] M. Gonzalez Harbour J. Palencia Gutierrez. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, 1998.
[7] M. Joseph and P.K. Pandya. Finding response times in a real-time system. In *The Computer Journal*, 1986.
[8] C. Kenna, J. Herman, B. Brandenburg, A. Mills, and J. Anderson. Soft real-time on multiprocessors: Are analysis-based schedulers really worth it? In *RTSS*, 2011.
[9] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, 1990.
[10] V. Pollex, S. Kollmann, and F. Slomka. Generalizing response-time analysis. In *RTCSA*, 2010.
[11] K. Richter. Compositional scheduling analysis using standard event models. In *Ph.D. thesis, Technical University of Braunschweig*, 2004.
[12] J. Rox and R. Ernst. Compositional performance analysis with improved analysis techniques for obtaining viable end-to-end latencies in distributed embedded systems. In *STTT*, 2013.
[13] I. Shin and I. Lee. Compositional real-time scheduling framework. In *RTSS*, 2004.
[14] Macro Spuri. Analysis of deadline scheduled real-time systems. In *RR-2772, INRIA, France*, 1996.
[15] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. In *Microprocessing and Microprogramming*, 1994.