2014 IEEE Real-Time Systems Symposium

Approximate Response Time Analysis of Real-Time Task Graphs

Nan Guan[1,2], Chuancai Gu[1], Martin Stigge[2], Qingxu Deng[1] and Wang Yi[2]

[1] Northeastern University, China
[2] Uppsala University, Sweden

***Abstract*—The response time analysis problem is intractable for most existing real-time task models, except the simplest ones. Exact solutions for this problem in general have exponential complexity, and may run into scalability problems for large-scale task systems. In this paper, we study *approximate* analysis for static-priority scheduling of the Digraph Real-Time task model, which is a generalization of most existing graph-based real-time task models. We present two approximate analysis methods RBF and IBF, both of which have pseudo-polynomial complexity. We quantitatively evaluate their analysis precision using the metric *speedup factor*. We prove that RBF has a speedup factor of 2, and this is tight even for dual-task systems. The speedup factor of IBF is an increasing function with respect to k, the number of interfering tasks. This function converges to 2 as k approaches infinity and equals 1 when $k = 1$, implying that the IBF analysis is exact for dual-task systems. We also conduct simulation experiments to evaluate the precision and efficiency of RBF and IBF with randomly generated task sets. Results show that the proposed approximate analysis methods have very high efficiency with low precision loss.**

I. Introduction

Traditionally, real-time task systems are modeled as collections of periodically repeating computational requests [9], [1]. However, behaviors that are not entirely periodic cannot be expressed accurately with this simple periodic task model. Examples include variable rate-dependent behavior in controllers for fuel injection in combustion engines [6] and frame dependent execution times in video codecs [11]. A natural representation of these complex structures is a task graph. Over years, more and more expressive graph-based task models are proposed to precisely describe complex embedded real-time systems [11], [5], [2], [3], [4], [14], [15].

Unfortunately, response time analysis of static-priority (SP) scheduling is intractable for most existing real-time task models, except the simplest ones. Figure 1 summarizes the time complexity of response time analysis for some common task models, as well as the generalization relations among them [13]. Pseudo-polynomial solutions only exist for the very simple L&L [9] and sporadic [1] task models, and the problem becomes strongly *co*NP-hard as soon as branching or phasing is allowed [17]. Algorithms with aggressive optimization techniques [18], [16] have been proposed to prune away a significant portion of state-space to be explored. However, these algorithms are still with exponential time complexity in general and may run into scalability problems with large-scale systems.

In this paper we study *approximate* response time analysis for real-time task graph models. In particular, we assume the *Digraph Real-Time* (DRT) task model [14], as it is a generalization of most existing graph-based real-time task models. We present two approximate response time analysis methods RBF and IBF, both with pseudo-polynomial time complexity and can handle large-scale task systems in very short time.

The main theoretical contribution of this paper is to provide quantitative performance guarantees for the proposed methods using the metric *speedup factor* [8], which is widely used in performance analysis of approximate algorithms for many scheduling problems. Our main results can be summarized as follows:

1) RBF has a speedup factor of 2, and this is tight, even for dual-task systems.
2) IBF has a speedup factor of $1 + \frac{\sqrt{k^2-k}}{k}$, where k is the number of interfering tasks (tasks with priorities higher than the one under analysis).

As a direct implication of 2), IBF turns out to be an exact analysis method for dual-task systems, since $1 + \frac{\sqrt{k^2-k}}{k} = 1$ when $k = 1$. Moreover, since $1 + \frac{\sqrt{k^2-k}}{k}$ is a monotonically increasing function with respect to k, the pessimism of IBF increases as the number of interfering tasks increases. As k approaches infinity, the speedup factor of IBF converges to 2, which is the same as RBF.

We also conduct simulation experiments to evaluate the analysis precision and efficiency of RBF and IBF. Experiment results show that our proposed approximate analysis methods (especially IBF) have precision rather close to the exact analysis, and confirm the trend that IBF is more precise for task sets with fewer tasks as indicated by its speedup factor. On the other hand, the analysis efficiency of the approximate analysis methods are very high. The analysis with both RBF and IBF finishes in at most several seconds for large-scale task systems (for which the exact analysis may take hours).

A. Related Work

Much work has been done to study the expressiveness and analysis efficiency of different task models, as summarized in Figure 1 (taken from [13]). Except for the very simple L&L and sporadic task models, the response time analysis problem of all these models are strongly *co*NP-hard. Although the proposed approximate analysis methods in this paper are

1052-8725/14 $31.00 © 2014 IEEE
DOI 10.1109/RTSS.2014.20

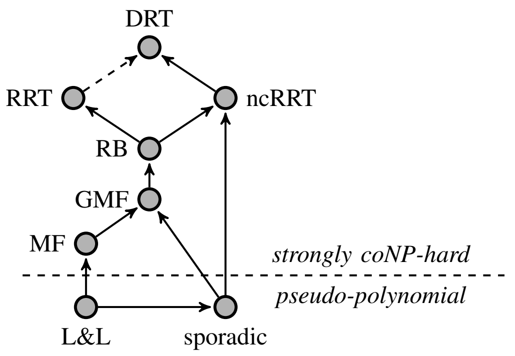


Figure 1. Classification of task models into tractable (pseudo-polynomial solutions exist) versus intractable (strongly coNP-hard) for the response time analysis problem, taken from [13]. The L&L is Liu and Layland's implicit-deadline periodic task model [9]; the sporadic task model is proposed in [1]; MF is the Multi-frame task model [11]; GMF is the Generalized Multi-frame task model [5]; RB is the Recurring Branching task model [2]; RRT is the Recurring Real-time task model [3]; ncRRT is the non-cyclic Recurring Real-time task model [4]; DRT is the Digraph Real-Time task model [14].

presented in the context of the DRT task model, they are directly applicable to other more restricted task models in Figure 1. The speedup factors derived in this paper also hold for other models in the figure.

The approximate analysis methods in this paper are closely related to the pseudo-polynomial algorithm in [14] for EDF scheduling analysis of DRT tasks (which is a tractable problem in contrast to the SP scheduling analysis problem studied in this paper). The IBF analysis proposed in this paper uses the same idea for workload abstraction as in [19], which aims at analyzing a more restricted GMF (generalized multi-frame) model. The method in [19] can be viewed as a special case of the IBF analysis proposed in this paper. It was claimed in [19] that using this abstraction gives an exact solution with pseudo-polynomial complexity for SP scheduling analysis of GMF, which is actually *not* true as pointed out in [17]. Indeed, the analysis for SP scheduling of GMF is strongly *co*NP-hard [17], so no pseudo-polynomial solution exists unless P=NP. To the best of our knowledge, no existing work has been done on the approximate analysis of any of these task graph models with quantitative performance guarantees.

The offset-based task model [20], [7] is widely used in the modeling and analysis of distributed real-time systems. This model has a close relation to task graph models in Figure 1. The reduction techniques used in [17] can be easily adapted to prove that the SP scheduling analysis problem of offset-based tasks is also *co*NP-hard in the strong sense. [10] proposed a similar workload abstraction as in [19] (and also the one used by IBF) for approximate analysis of offset-based tasks. However, only experimental evaluations are conducted but no quantitative performance guarantee is provided in [10]. Our major theoretical contribution over the work in [19] and [10] is to quantify performance guarantee of different workload abstractions when finding the exact critical instant is intractable due to combinatorial explosion. Actually, the speedup factor results of this paper also hold for the analysis proposed in [10] with the offset-based task model.

II. Diagraph Task Model

A. Syntax

A task set consists of N independent tasks $\{T_1, \cdots, T_N\}$. A task T is represented by a directed graph $G(T) = (V(T), E(T))$ with $V(T)$ denoting the set of vertices and $E(T)$ the set of edges of the graph.

The vertices $V(T) = \{v_1, \cdots, v_n\}$ represent the types of all jobs that can be released by T. Each vertex v is labeled with a tuple $\langle e(v), d(v) \rangle$, where $e(v) \in \mathbb{N}$ denotes the worst-case execution time (WCET), $d(v) \in \mathbb{N}$ denotes the relative deadline. We implicitly assume the relation $e(v) \leq d(v)$ for all job types v.

The edges of $G(T)$ represent the order in which jobs generated by T are released. Each edge $(u, v) \in E(T)$ is labeled with $p(u, v) \in \mathbb{N}$ denoting the minimum inter-release separation time between u and v. The general DRT task model in [14] does not regulate the relation between the deadline of a vertex and the inter-release separations marked on the outgoing edges of the vertex, but in this paper we assume deadlines to be constrained, i.e., for each vertex u we have $d(u) \leq p(u, v)$ for all edges.

B. Semantics

A job J is represented by a tuple (r, e) consisting of an absolute release time r and an execution time e. The semantics of a DRT task system is defined as the set of *job sequences* it may generate: $\sigma = [(r_0, e_0), (r_1, e_1), ...]$ is a job sequence if all jobs are monotonically ordered by release times, i.e., $r_i \leq r_j$ for $i \leq j$. A job sequence $\sigma = [(r_0, e_0), (r_1, e_1), ...]$ is generated by T if $\pi = (v_0, v_1, \cdots)$ is a path in $G(T)$ and for all $i \geq 0$:

1) $r_{i+1} - r_i \geq p(v_i, v_{i+1})$ and
2) $e_i \leq e(v_i)$

Combining the job sequences of individual tasks results in a job sequence of the task set.

Example 1. *Figure 2 shows an example to illustrate the semantics of DRT tasks. When the system starts, T releases its first run-time job by an arbitrary vertex. Then the released sequence corresponds to a particular direct path through $G(T)$. Consider the job sequence $\sigma = [(2,3), (10,1), (25,4), (37,1)]$ which corresponds to path $\pi = (v_1, v_2, v_3, v_2)$ in $G(T)$. Note that this example demonstrates the "sporadic" behavior allowed by the semantics of the DRT model. The first job in σ (corresponds to v_1) is released at time 2, and the second job in σ (v_2) is released 2 time units later than its earliest possible release time, while the job of v_3 and the second job of v_2 are released as early as possible.*

C. SP Scheduling and Worst-Case Response Time

We use the static-priority (SP) scheduling algorithm to schedule jobs released by all tasks. Each task is assigned a static priority in a priori, and each of its released job inherits this priority. At each time instant during runtime, the job with the highest priority among all the jobs that have been released but not finished yet is selected for execution.

Definition 1 (Worst-Case Response Time)**.** *Given a task set τ, the* worst-case response time *of vertex $v \in V(T)$ for a task*

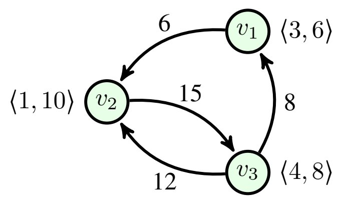


Figure 2. An example task containing three different job types.

$T \in \tau$ is the maximal time between release and finish of any job corresponding to v in all job sequences generated by τ.

A vertex v is *schedulable* if its worst-case response time is no larger than its relative deadline $d(v)$. The worst-case response time of a vertex is not affected by the workload of tasks with lower priorities. Moreover, it is not affected by other vertices from the same task assuming that other vertices are all schedulable (since $d(u) \le p(u,v)$ for all edges $(u,v) \in E(T)$). Thus, we can analyze the worst-case response time of each vertex v independently under the interference of higher-priority task set τ. We use $R(v,\tau)$ to denote the worst-case response time of v with *interfering task set* τ.

III. Approximate Response Time Analysis

This section introduces two *approximate* response time analysis methods RBF and IBF, both of which have *pseudo-polynomial* complexity. Before presenting them, we first briefly review some basic concepts and the exact response time analysis method with *exponential* complexity [18], [16].

The workload of a path π can be abstracted with a *request function* [18], which for each t returns the maximal accumulated execution requirement of all jobs that π may release until time t (suppose the first job of π is released at time 0).

Definition 2 (Request Function)**.** *For a path $\pi = (v_0, \cdots, v_l)$ through the graph $G(T)$ of a task T, we define its* request function *as*

$$rf_\pi(t) := \max\{\, e(\pi') \mid \pi' \textit{ is prefix of } \pi \textit{ and } p(\pi') < t \,\}$$

where $e(\pi) := \sum_{i=0}^{l} e(v_i)$ and $p(\pi) := \sum_{i=0}^{l-1} p(v_i, v_{i+1})$.

$rf_\pi(t)$ is a non-decreasing staircase function with respect to t. Each horizontal segment is left-open and right-closed. In particular, $rf_\pi(0) = 0$.

Let $\Pi(T)$ denote the set of paths in $G(T)$ and $\Pi(\tau) := \Pi(T_1) \times \Pi(T_2) \times \cdots \times \Pi(T_n)$, i.e., the set of all combinations of paths from tasks in task set $\tau = \{T_1, T_2, \cdots, T_n\}$. Further, let $\bar{\pi} = (\pi_1, \pi_2, \cdots, \pi_n)$ denote an element of $\Pi(\tau)$, i.e., a single combination of paths of different tasks. If we are analyzing the response time of vertex v with interfering task set τ, then the *total request function* of v with a path combination $\bar{\pi}$ is

$$rf_{(v,\bar{\pi})}(t) = e(v) + \sum_{\pi_i \in \bar{\pi}} rf_{\pi_i}(t)$$

We can express the response time of a vertex v with this particular path combination as:

$$R(v,\bar{\pi}) = \min_{t>0} \left\{ t \;\middle|\; rf_{(v,\bar{\pi})}(t) \le t \right\} \tag{1}$$

The overall worst-case response time of v with interfering task set τ is the maximum over all path combinations:

$$R(v,\tau) = \max_{\bar{\pi} \in \Pi(\tau)} \{R(v,\bar{\pi})\} \tag{2}$$

The number of path combinations in $\Pi(\tau)$ is exponential and using (2) to compute the exact worst-case response time $R(v)$ has an exponential complexity. The exponential explosion comes from two sources: (1) the number of paths in each task, and (2) the number of combinations of paths from different tasks. A refinement-based strategy [18] has been developed to rule out a significant portion of path combinations in the computation using (2), but it still has exponential complexity in general and may run into scalability problems with large-scale task systems. Indeed, computing the exact worst-case response time is *co*NP-hard in the strong sense [17], so no (pseudo-)polynomial solution exists unless P=NP.

In the following, we present two approximate response time analysis methods, both with pseudo-polynomial complexity. The idea is to use abstractions to over-approximate the workload of each task, such that the workload approximation of each task and the workload composition of different tasks both can be performed efficiently.

A. RBF*: Analysis by Request Bound Functions*

Our first approximate analysis method RBF is based on the abstraction *Request Bound Function*:

Definition 3 (Request Bound Function)**.** *For a task T, we define its* request bound function *$rbf_T(t)$ as*

$$rbf_T(t) := \max_{\pi \in \Pi(T)} \{rf_\pi(t)\}$$

The total request bound function *for a vertex v under analysis with interfering task set τ is*

$$rbf_{(v,\tau)}(t) := e(v) + \sum_{T \in \tau} rbf_T(t)$$

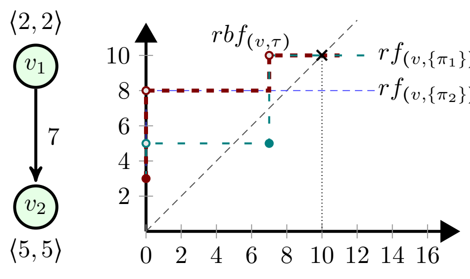
Theorem 1 (RBF Analysis)**.** *Given a vertex v with interfering task set τ, $R_{\mathsf{RBF}}(v,\tau)$ is a safe upper bound of its worst-case response time:*

$$R(v,\tau) \le R_{\mathsf{RBF}}(v,\tau) := \min_{t>0} \left\{ t \;\middle|\; rbf_{(v,\tau)}(t) \le t \right\} \tag{3}$$

Proof: By the definition of $rbf_{(v,\tau)}(t)$ we know $rbf_{(v,\tau)}(t) \ge rf_{(v,\bar{\pi})}(t)$ for any path combination $\bar{\pi}$, which implies $R_{\mathsf{RBF}}(v,\tau) \ge \max_{\bar{\pi} \in \Pi(\tau)}\{R(v,\bar{\pi})\} = R(v,\tau)$. ■

Example 2. *Consider a vertex v with $e(v) = 3$ and interfering task set τ containing only one task T, as shown in Figure 3-(a). $G(T)$ consists of two vertices v_1 and v_2 connected by an edge (v_1, v_2), with $e(v_1) = 2$, $e(v_2) = 5$ and $p(v_1, v_2) = 5$. $G(T)$ has two paths $\pi_1 = (v_1, v_2)$ and $\pi_2 = (v_2)$. $rf_{(v,\{\pi_1\})}(t)$ and $rf_{(v,\{\pi_2\})}(t)$ intersects with the diagonal at $t = 5$ and $t = 8$, respectively, so the worst-case response time of v is 8. However, $rbf_{(v,\tau)}(t)$ intersects with the diagonal at $t = 10$ and thus $R_{\mathsf{RBF}}(v,\tau) = 10$.*

In (3), the individual request bound functions of tasks are summed up to over-approximate the total computational requests of tasks in τ, which solves the second source of



(a) T (b) $R_{\mathsf{RBF}}(v,\tau)$

Figure 3. Illustration of computing $R_{\mathsf{RBF}}(v,\tau)$ by $rbf_{(v,\tau)}$.

```
Ω_0 ← {⟨e(u), 0, u⟩ | u ∈ G(T)}
for k=1 to t do
    Ω_k = ∅
    for all ⟨e, r, u⟩ ∈ Ω_{k−1} do
        for all edges (u, w) ∈ G(T) do
            e′ ← e + e(w)
            r′ ← r + p(u, w)
            if r′ ≤ t then
                Ω_k ← Ω_k ∪ {⟨e′, r′, w⟩}
            end if
        end for
    end for
end for
return max { e | ⟨e, r, u⟩ ∈ ⋃_{k≤t} Ω_k }
```

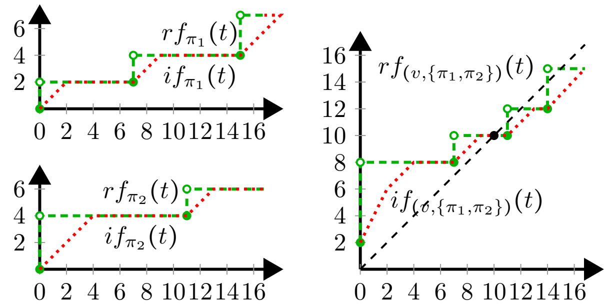
Figure 4. Algorithm for computing $rbf_T(t)$.

the exponential explosion, i.e., the combination of paths from different tasks. The first source of the exponential explosion, the number of paths of each individual task, can be tamed by the *path abstraction* technique proposed in [14]. We can use the path abstraction technique to compute $rbf_{T_i}(t)$ for each task T_i with $O(t^2 n)$ time complexity where n is the number of vertices in $G(T_i)$. Figure 4 shows the pseudo-code of the algorithm for computing $rbf_{T_i}(t)$. We only need to compute $rbf_{T_i}(t)$ for t up to $d(v)$ to decide whether a vertex is schedulable or not. So the overall time complexity of RBF is pseudo-polynomial. Several optimizations [14] can be applied to significantly improve the efficiency of the algorithm in Figure 4. We refer to [14] for details of these optimizations, as well as the intuition of the path abstraction technique. Note that $rbf_{T_i}(t)$ is a left-open and right-closed staircase function. By this algorithm we get a left-closed and right-open staircase function, but it can be easily transferred into a corresponding left-open and right-closed function in polynomial time with respect to t.

B. IBF*: Analysis by Interference Bound Functions*

Our second approximate analysis IBF uses a more precise abstraction *interference bound function* to represent the workload of each task. We first introduce the *interference function* of individual paths and path combinations:

Definition 4 (Interference Function)**.** *For a path* $\pi = (v_0, \cdots, v_l)$ *through the graph* $G(T)$ *of a task* T*, we define*



(a) rf and if of π_1 and π_2 (b) rf and if of $(v,\{\pi_1,\pi_2\})$

Figure 5. Illustration of the different between rf and if.

its interference function as

$$if_\pi(t) := \max\{\, ee(\pi') \mid \pi' \text{ is prefix of } \pi \text{ and } p(\pi') < t \,\}$$

where $p(\pi) := \sum_{i=0}^{l-1} p(v_i, v_{i+1})$ *and*

$$ee(\pi) := \sum_{i=0}^{l-1} e(v_i) + \min\left(e(v_l), \max(0, t - p(\pi))\right)$$

Given vertex v *under analysis and a path combination* $\bar{\pi}$ *generated by the interfering task set, the* total interference function *for* v *and* $\bar{\pi}$ *is*

$$if_{(v,\bar{\pi})}(t) = \sum_{\pi_i \in \bar{\pi}} if_{\pi_i} + e(v)$$

The interference function $if_\pi(t)$ of a single path is a slanted staircase function. The slope of each slanted segments in $if_\pi(t)$ is either 0 or 1. $if_{(v,\bar{\pi})}(t)$ is also a slanted staircase function, but it may contain slanted segments with slope greater than 1 when if_π of several paths increases at the same time. Figure 5 illustrates the interference function of two paths π_1 and π_2, and the total interference function of path combination $\{\pi_1, \pi_2\}$ and a vertex v with $e(v) = 2$.

Actually, $if_{(v,\bar{\pi})}$ and $rf_{(v,\bar{\pi})}$ are identical if they are used for calculating the *exact* response time by enumerating all path combinations, as stated in the following theorem:

Theorem 2. *The exact worst-case response time of* v *with interfering path combination* $\bar{\pi}$ *is computed by*

$$R(v,\bar{\pi}) = \min_{t>0} \left\{ t \;\middle|\; if_{(v,\bar{\pi})}(t) \le t \right\} \tag{4}$$

and the total worst-case response time of v *is computed by:*

$$R(v,\tau) = \max_{\bar{\pi} \in \Pi(\tau)} \{R(v,\bar{\pi})\} \tag{5}$$

To see the correctness of the theorem, it is sufficient to show that the worst-case response time of v with a particular path combination $\bar{\pi}$ computed using $if_{(v,\bar{\pi})}(t)$ and $rf_{(v,\bar{\pi})}(t)$ are the same, i.e., the RHS of (1) equals the RHS of (4). By checking the definitions of $if_{(v,\bar{\pi})}(t)$ and $rf_{(v,\bar{\pi})}(t)$, it is easy to see that they overlap with each other in the horizontal segments of $if_{(v,\bar{\pi})}(t)$. Moreover, since each segment of $if_{(v,\bar{\pi})}(t)$ is either horizontal, or has a slope at least 1, both $if_{(v,\bar{\pi})}(t)$ and $rf_{(v,\bar{\pi})}(t)$ intersect with the diagonal only in the horizontal segments of $if_{(v,\bar{\pi})}(t)$, as illustrated by Figure 5. Therefore we can conclude that $if_{(v,\bar{\pi})}(t)$ and $rf_{(v,\bar{\pi})}(t)$

intersect with the diagonal at exactly the same points, and thus the worst-case response time of v with a particular path $\bar{\pi}$ computed using $if_{(v,\bar{\pi})}(t)$ and $rf_{(v,\bar{\pi})}(t)$ are the same. A formal proof is omitted here.

Again, enumerating the interference functions for all path combinations is computationally intractable. In the following we introduce the *interference bound function* abstraction and the corresponding approximate analysis method. The *interference bound function* is defined in a way analog to the *request bound function*, but results in better analysis precision.

Definition 5 (Interference Bound Function)**.** *For a task T, we define its interference bound function $ibf_T(t)$ as*

$$ibf_T(t) := \max_{\pi \in \Pi(T)} \{if_\pi(t)\}$$

The total interference bound function *for a vertex v under analysis with interfering task set τ is*

$$ibf_{(v,\tau)}(t) := e(v) + \sum_{T \in \tau} ibf_T(t)$$

Theorem 3 (IBF Analysis)**.** *Given a vertex v with interfering task set τ, $R_{\mathsf{IBF}}(v,\tau)$ is a safe upper bound of its worst-case response time:*

$$R(v,\tau) \leq R_{\mathsf{IBF}}(v,\tau) := \min_{t>0} \left\{ t \;\middle|\; ibf_{(v,\tau)}(t) \leq t \right\} \quad (6)$$

Proof: $ibf_{T_i}(t)$ over-approximates $if_{\pi_i}(t)$ for any path $\pi_i \in G(T_i)$, so $R_{\mathsf{IBF}}(v,\tau)$ over-approximates $R(v,\tau)$ computed by $if_{(v,\bar{\pi})}(t)$ enumerating all path combinations. ■

The algorithm in Figure 4 can be reused to compute $ibf_{(v,T)}(t)$ for each task T in pseudo-polynomial time, with the last line modified as:

"**return** $\max\{\, e + \min(e(u), t - r) \mid \langle e, p, u\rangle \in \bigcup_{k \leq t} DT_k\}$"

The IBF analysis method strictly dominates RBF, denoted by IBF $\succ$ RBF:

Theorem 4 (IBF $\succ$ RBF)**.** *For any vertex v it holds $R_{\mathsf{IBF}}(v,\tau) \leq R_{\mathsf{RBF}}(v,\tau)$ and there exists some v for which it holds $R_{\mathsf{IBF}}(v,\tau) < R_{\mathsf{RBF}}(v,\tau)$.*

Proof: $R_{\mathsf{IBF}}(v,\tau) \leq R_{\mathsf{RBF}}(v,\tau)$ holds since if is a more precise abstraction than rf. The task system in Example 2 witnesses $R_{\mathsf{IBF}}(v,\tau) < R_{\mathsf{RBF}}(v,\tau)$. Figure 6 shows $R_{\mathsf{IBF}}(v,\tau) = 8$, which is smaller than $R_{\mathsf{RBF}}(v,\tau) = 10$. ■

For the task set in Example 2, $R_{\mathsf{IBF}}(v,\tau)$ equals v's exact worst-case response time $R(v,\tau)$. However, in general $R_{\mathsf{IBF}}(v,\tau)$ is an over-approximation of $R(v,\tau)$. An example in which $R_{\mathsf{IBF}}(v,\tau)$ is *strictly* larger than $R(v,\tau)$ can be found in [17].

C. Some Properties

We use RF and IF to denote the exact response time analysis using the request function rf and interference function if respectively. Then we know the following relations:

$$\begin{array}{ccc} \mathsf{IF} & \succ & \mathsf{IBF} \\ \shortparallel & & \curlyvee \\ \mathsf{RF} & \succ & \mathsf{RBF} \end{array}$$

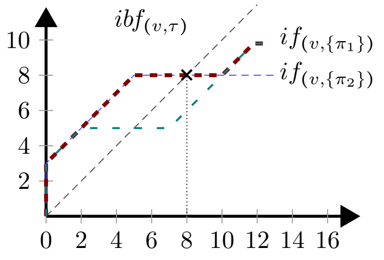


Figure 6. Illustration of $ibf_{(v,\tau)}$.

where "$\succ$" denotes the *strict domination* relation introduced in above.

Before ending this section, we introduce some auxiliary lemmas, which are useful in the next section for deriving the speedup factors of RBF and IBF.

First, we shall clarify that for each particular t, $rbf_{(v,\tau)}(t)$ and $ibf_{(v,\tau)}(t)$ do *not* introduce any pessimism as they indeed capture the exact worst-case workload (interference) incurred by some path combinations in a time interval of length t, as stated in the following lemma:

Lemma 1. *Given a vertex v and interfering task set τ, for any $t \geq 0$ it holds:*

$$rbf_{(v,\tau)}(t) = \max_{\bar{\pi} \in \Pi(\tau)} \{rf_{(v,\bar{\pi})}(t)\} \quad (7)$$

$$ibf_{(v,\tau)}(t) = \max_{\bar{\pi} \in \Pi(\tau)} \{if_{(v,\bar{\pi})}(t)\} \quad (8)$$

Proof: We prove (7), and (8) can be proved similarly. First it is easy to see the relation $\geq$ between the LHS and RHS of (7). In the following we show the relation $\leq$ also holds. We start with the definition of $rbf_{(v,\tau)}(t)$:

$$\begin{aligned} rbf_{(v,\tau)}(t) &= rbf_{T_1}(t) + \cdots + rbf_{T_n}(t) + e(v) \\ &= \max_{\pi \in \Pi(T_1)} \{rf_\pi(t)\} + \cdots + \max_{\pi \in \Pi(T_n)} \{rf_\pi(t)\} + e(v) \end{aligned}$$

Let π_i be the path in $\Pi(T_i)$ s.t. $rf_{\pi_i}(t) \geq rf_\pi(t)$ for all $\pi \in \Pi(T_i)$, then

$$\begin{aligned} rbf_{(v,\tau)}(t) &= rf_{\pi_1}(t) + \cdots + rf_{\pi_n}(t) + e(v) \\ &= rf_{(v,\{\pi_1 \cdots \pi_n\})}(t) \\ &\leq \max_{\bar{\pi} \in \Pi(\tau)} \{rf_{(v,\bar{\pi})}(t)\} \end{aligned}$$

■

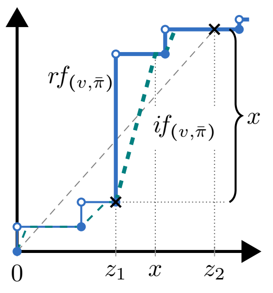
Definition 6 (Lifting Point)**.** *We say z is a* lifting point *of a (slanted) staircase function $f(t)$ iff*

$$f(z) = f(z - \varepsilon) \;\wedge\; f(z) < f(z + \varepsilon)$$

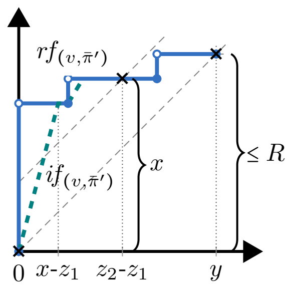
where ε is a positive number arbitrarily close to 0.

Intuitively, a lifting point is the x-axis value of the ending point of a horizontal segment. In Figure 7-(a), z_1 is a lifting point of both $rf_{(v,\bar{\pi})}(t)$ and $if_{(v,\bar{\pi})}(t)$.

Lemma 2. *Given a vertex v and the interfering task set τ. Let $\bar{\pi}$ be an arbitrary path combination in $\Pi(\tau)$. Let $f(t)$ be*



(a) $rf_{(v,\bar{\pi})}(t)$ and $if_{(v,\bar{\pi})}(t)$



(b) $rf_{(v,\bar{\pi}')}(t)$ and $if_{(v,\bar{\pi}')}(t)$

Figure 7. Intuition of Lemma 2

$rf_{(v,\bar{\pi})}(t)$ *or* $if_{(v,\bar{\pi})}(t)$ *Suppose* $f(x) > x$*, and define*

$$z_2 := \min_{t>x}\{\, t \mid f(t) \leq t \,\}$$
$$z_1 := \max_{t<x}\{\, t \mid f(t) \leq t \wedge t \text{ is a lifing point of } f(t) \,\}$$

then it holds

$$f(z_2) - f(z_1) \leq R(v,\tau)$$

The proof of Lemma 2 is provided in the appendix. The intuition of the lemma is shown in Figure 7. The part of $rf_{(v,\bar{\pi})}(t)$ between z_1 and z_2 can be viewed as $rf_{(v,\bar{\pi}')}(t)$ with another path combination $\bar{\pi}'$, in which each element is the suffix of the counter part in $\bar{\pi}$ starting from z_1, i.e., $rf_{(v,\bar{\pi})}(z_2) - rf_{(v,\bar{\pi})}(z_1) = rf_{(v,\bar{\pi}')}(z_2 - z_1)$. Suppose $rf_{(v,\bar{\pi}')}(t)$ intersects with the diagonal at y, then we know $rf_{(v,\bar{\pi}')}(z_2 - z_1) \leq rf_{(v,\bar{\pi}')}(y) = y \leq R(v,\tau)$ since the response time of v with any particular path combination is bounded by $R(v,\tau)$, and thus $rf_{(v,\bar{\pi})}(z_2) - rf_{(v,\bar{\pi})}(z_1)$ is also bounded by $R(v,\tau)$. The intuition is similar for the case of $f(t) = if_{(v,\bar{\pi})}(t)$.

IV. Speedup Factor

We quantify the performance guarantee of RBF and IBF by the metric *speedup factor* [8]:

Definition 7 (Speedup Factor)**.** *A method M has a* speedup factor *of s if the response time estimation returned by M on a speed-s processor for any vertex in any task system is bounded by its exact worst-case response time.*

The speedup factor s is tight *if on any speed-s' processor where $s' < s$, there exists some task system in which a vertex's response time estimation returned by M is larger than its exact response time on a speed-1 processor.*

If a job executes for $e(v)$ on a speed-1 machine, then it finishes the same amount of workload in $e(v)/s$ time on a speed-s machine. So we can reload the request function and interference function on a speed-s processor as

$$rf^s_\pi(t) := \max\{\, e^s(\pi') \mid \pi' \text{is prefix of } \pi \text{ and } p(\pi') < t \,\}$$

$$if^s_\pi(t) := \max\{\, ee^s(\pi') \mid \pi' \text{is prefix of } \pi \text{ and } p(\pi') < t \,\}$$

where $p(\pi) := \sum_{i=0}^{l-1} p(v_i, v_{i+1})$ and

$$e^s(\pi) := \sum_{i=0}^{l} e(v_i)/s$$
$$ee^s(\pi) := \sum_{i=0}^{l-1} e(v_i)/s + \min\left(e(v_l)/s, \max(0, t - p(\pi))\right)$$

The notations $rf^s_{(v,\bar{\pi})}(t)$, $rbf^s_T(t)$, $rbf^s_{(v,\tau)}(t)$, $if^s_{(v,\bar{\pi})}(t)$, $ibf^s_T(t)$ and $ibf^s_{(v,\tau)}(t)$ are defined in the same way as their counterpart in the last section, but based on $rf^s_\pi(t)$, $if^s_\pi(t)$ and $e(v)/s$. For simplicity, we omit the subscript s in the above notations when $s = 1$.

It is easy to see the following relations:

$$rf^s_\pi(t) = rf_\pi(t)/s$$
$$rf^s_{(v,\bar{\pi})}(t) = rf_{(v,\bar{\pi})}(t)/s$$
$$rbf^s_T(t) = rbf_T(t)/s$$
$$rbf^s_{(v,\tau)}(t) = rbf_{(v,\tau)}(t)/s$$

However, the corresponding relations do not always hold for interference (bound) functions. They only hold for the points on horizonal segments of these functions on speed-1 processors:

Lemma 3. *Given a total interference function $if_{(v,\bar{\pi})}(t)$, for any point x that is not differentiable or satisfies $\mathsf{d}if_{(v,\bar{\pi})}(x)/\mathsf{d}(x) = 0$ (i.e., the points on the horizontal segments of $if_{(v,\bar{\pi})}(t)$), it holds*

$$if^s_{(v,\bar{\pi})}(x) = if_{(v,\bar{\pi})}(x)/s \tag{9}$$

The lemma can be easily proved by checking the definition of $if^s_{(v,\bar{\pi})}$. Note that the analog conclusions also hold for $if^s_\pi(t)$, $ibf^s_T(t)$ and $ibf^s_{(v,\tau)}(t)$. In Figure 8, the points in $[4,14]$ and $[16,18]$ satisfy (9).

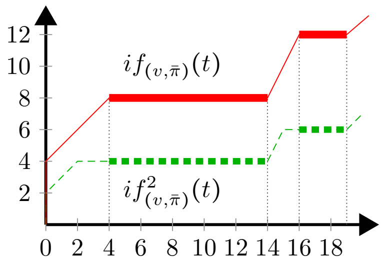


Figure 8. $if^2_{(v,\bar{\pi})}(t)$ and $if_{(v,\bar{\pi})}(t)$.

A. Speedup Factor of RBF

RBF has a *tight* speedup factor of 2 as proved in the following two theorems.

Theorem 5. RBF *has a speedup factor of* 2.

Proof: Let $R = R(v,\tau)$. We shall prove that if v's worst-case response time on a speed-1 processor is R, then the response time estimation by RBF is bounded by R on a speed-2 processor. We will show that $\forall \bar{\pi} \in \Pi(\tau)$ it holds

$$rf^2_{(v,\bar{\pi})}(R) \leq R \tag{10}$$

If this is true, we have $\max_{\bar{\pi}\in\Pi(\tau)}\{rf^2_{(v,\bar{\pi})}(R)\} \le R$, and by Lemma 1 we know $rbf^2_{(v,\tau)}(R) \le R$, which completes the proof. Since (10) trivially holds if $rf_{(v,\bar{\pi})}(R) \le R$, in the rest of the proof we focus on the interesting case of $rf_{(v,\bar{\pi})}(R) > R$. We define:

$$z_2 := \min_{t>R}\{\, t \mid rf_{(v,\bar{\pi})}(t) \le t \,\}$$
$$z_1 := \max_{t<R}\{\, t \mid rf_{(v,\bar{\pi})}(t) \le t \wedge t \text{ is a lifing point of } rf_{(v,\bar{\pi})}(t) \,\}$$

(We refer to Figure 7-(a) for illustration of the definitions of z_1 and z_2 with $x = R$.)

Thus, by Lemma 2 we know

$$rf_{(v,\bar{\pi})}(z_2) - rf_{(v,\bar{\pi})}(z_1) \le R$$

and since $rf_{(v,\bar{\pi})}(z_1) \le z_1 < R$, we know

$$rf_{(v,\bar{\pi})}(z_2) \le 2\cdot R \tag{11}$$

Since $R < z_2$, we have $rf_{(v,\bar{\pi})}(R) \le rf_{(v,\bar{\pi})}(z_2)$, combining with (11) we have

$$rf_{(v,\bar{\pi})}(R) \le 2\cdot R$$

Applying the relation $rf^2_{(v,\bar{\pi})}(R) = rf_{(v,\bar{\pi})}(R)/2$ to the above inequality we finally have $rf^2_{(v,\bar{\pi})}(R) \le R$. ■

Theorem 6. *The the speedup factor* 2 *of* RBF *is tight, even for dual-task systems.*

Proof: We prove the theorem by showing that for any $s = 2-\varepsilon$ where $1 > \varepsilon > 0$, a dual-task system $\tau = \{T_1, T_2\}$ can be constructed such that a vertex v's exact worst-case response time on a speed-1 processor is r, but the response time bound estimation returned by RBF on a speed-s processor is strictly larger than r. The construction is as follows:

- Let $r = \lceil 2/\varepsilon \rceil + 2$, and $k = \lceil r/2 \rceil$.
- The higher priority task T_1 consists of $k+1$ vertices $\{v_0, v_1, \cdots, v_k\}$. For each vertex v_i, let $e(v_i) = r-(i+1)$ and $d(v_i) = r-i$. Each vertex v_i with $i \in [1,k]$ is connected to v_0 via an edge (v_i, v_0), and let $p(v_i, v_0) = r-i$.
- The lower priority task T_2 only has one vertex v, with $e(v) = 1$ and $d(v) = r$.

The task set is shown in Figure 9, where only the WCET of each vertex in T_1 is marked as they are all trivially schedulable and their deadlines are irrelevant to the response time of the vertex v in T_2.

We use π_i to denote the path in $G(T_1)$ starting with vertex v_i. It is easy to verify the worst-case response time of τ on a speed-1 processor: with each path $\pi_i \in G(T_1)$, v's response time is exactly $r-i$. In the following we will show that $rbf^s_{(v,\{T_1\})}(t) > t$ holds for any time point $t \in [0,r]$, which implies the response time estimation of v by the RBF analysis on a speed-s processor is strictly larger than r. We discuss two cases: (1) $t \in [0, r/2]$ and (2) $t \in (r/2, r]$.

1) $t \in [0, r/2]$. In this case, we have $rf_{(v,\pi_0)}(t) = (r-1)+1 = r$, and since $t \le r/2$ and $s < 2$ we know $rf_{(v,\pi_0)}(t) > s\cdot t$.
2) $t \in (r/2, d]$. In this case, there exists $i \in [1,k]$ satisfying $t \in (r-i, r-i+1]$. Consider path π_i. By $t > r-i$ we have

$$\begin{aligned} rf_{(v,\pi_i)}(t) &= e(v_i) + e(v_0) + 1 \\ &= 2\cdot r - i - 1 \\ &= 2\cdot(r-i+1) + (i-1) - 2 \\ &> 2\cdot(r-i+1) + (i-1)\cdot\varepsilon - r\cdot\varepsilon \\ &= (2-\varepsilon)\cdot(r-i+1) \\ &= s\cdot(r-i+1) \\ &\ge s\cdot t \end{aligned}$$

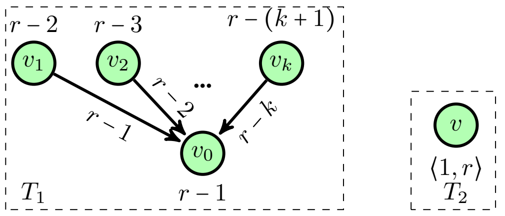


Figure 9. A task set of two tasks T_1 and T_2 showing the speedup factor 2 of RBF is tight.

In summary, in both cases there exists a path π in $\Pi(T_1)$ such that $rf_{(v,\pi)}(t) > s\cdot t$. On the other hand, we know

$$\begin{aligned} rbf^s_{(v,\{T_1\})}(t) &= rbf_{(v,\{T_1\})}(t)/s \\ &\ge rf_{(v,\pi)}(t)/s \end{aligned}$$

So $\forall t \in [0,r]$, it holds $rbf^s_{(v,\{T_1\})}(t) > t$, and thereby the response time estimation of v returned by RBF on a speed-s processor is strictly larger than r when $s < 2$. ■

B. Speedup Factor of IBF

We start with the fact that $if^s_{(v,\bar{\pi})}(t)$ is a Lipschitz continuous function [12] with a Lipschitz constant being the number of interfering tasks:

Lemma 4. *Given a vertex v and the interfering task set τ consisting of k tasks executing on a speed-s machine. Let $\bar{\pi}$ be an arbitrary path combination in $\Pi(\tau)$, then for any $x \ge y \ge 0$ it holds*

$$if^s_{(v,\bar{\pi})}(x) - if^s_{(v,\bar{\pi})}(y) \le k\cdot(x-y) \tag{12}$$

Proof: By the definition of $if^s_{(v,\bar{\pi})}$ we know

$$if^s_{(v,\bar{\pi})}(x) - if^s_{(v,\bar{\pi})}(y) = \sum_{\pi_i\in\bar{\pi}} (if^s_{\pi_i}(x) - if^s_{\pi_i}(y)) \tag{13}$$

By the definition of $if^s_{\pi}(t)$ we know

$$if^s_{\pi_i}(x) - if^s_{\pi_i}(y) \le x - y$$

i.e., the slope of slated segments of $if^s_{\pi_i}(t)$ is at most 1, and since τ consists of k tasks, we know $\sum_{\pi_i\in\bar{\pi}}(if^s_{\pi_i}(x) - if^s_{\pi_i}(y))$ is bounded by $k\cdot(x-y)$, which together with (13) concludes the lemma. ■

Note that speeding up the processor does not increase the Lipschitz constant of the interference function. Instead, it only decreases the length of the slanted segments, as shown in Figure 8.

Theorem 7. IBF *has a speedup factor of*

$$1 + \frac{\sqrt{k^2 - k}}{k}$$

for any vertex v with an interfering task set τ of k tasks.

Proof: Let $R = R(v, \tau)$. We shall prove that if v's exact worst-case response time on a speed-1 processor is R, then the response time upper bound returned by IBF on a speed-s processor is bounded by R, where $s \geq 1 + \frac{\sqrt{k^2-k}}{k}$. In the following we will show that for an arbitrary path combination $\bar{\pi} \in \Pi(\tau)$ it holds

$$if^s_{(v,\bar{\pi})}(R) \leq R \tag{14}$$

If this is true, we have

$$\max_{\bar{\pi} \in \Pi(\tau)} \{ if^s_{(v,\bar{\pi})}(R) \} \leq R$$

and by Lemma 1 we know $ibf^s_{(v,\tau)}(R) \leq R$, which completes the proof.

If $if_{(v,\bar{\pi})}(R) \leq R$, then (14) trivially holds. In the remained of the proof we focus on the interesting case of $if_{(v,\bar{\pi})}(R) > R$.

We define

$$z_2 := \min_{t > R} \{ t \mid if_{(v,\bar{\pi})}(t) \leq t \}$$

$$z_1 := \max_{t < R} \{ t \mid if_{(v,\bar{\pi})}(t) \leq t \wedge t \text{ is a lifting point of } if_{(v,\bar{\pi})}(t) \}$$

(We refer to Figure 7-(a) for illustration of the definitions of z_1 and z_2 with $x = R$.)

Thus, by Lemma 2 we know

$$if_{(v,\bar{\pi})}(z_2) - if_{(v,\bar{\pi})}(z_1) \leq R \tag{15}$$

We define $x = if_{(v,\bar{\pi})}(z_1)/R$, and discuss two cases: (i) $x \geq \frac{k \cdot s - s}{k \cdot s - 1}$ and (ii) $x < \frac{k \cdot s - s}{k \cdot s - 1}$:

1) $x \geq \frac{k \cdot s - s}{k \cdot s - 1}$. By Lemma 4 we have

$$if^s_{(v,\bar{\pi})}(R) - if^s_{(v,\bar{\pi})}(z_1) \leq k \times (R - z_1) \tag{16}$$

By the definition of z_1 we know z_1 is on a horizontal segment of $if_{(v,\bar{\pi})}(t)$, so by Lemma 3 and $if_{(v,\bar{\pi})}(z_1) \leq z_1$ we know

$$if^s_{(v,\bar{\pi})}(z_1) = if_{(v,\bar{\pi})}(z_1)/s \leq z_1/s \tag{17}$$

Combining (16) and (17) gives

$$if^s_{(v,\bar{\pi})}(R) \leq z_1/s + k \times (R - z_1) \tag{18}$$

On the other hand

$$\begin{aligned} & x \geq (k \cdot s - s)/(k \cdot s - 1) \\ \Rightarrow \; & x/s + k \cdot (1 - x) - 1 \leq 0 \\ \Rightarrow \; & if_{(v,\bar{\pi})}(z_1)/s + k \cdot (R - if_{(v,\bar{\pi})}(z_1)) \leq R \\ \Rightarrow \; & k \cdot R - if_{(v,\bar{\pi})}(z_1)(k - 1/s) \leq R \end{aligned}$$

and since $if_{(v,\bar{\pi})}(z_1) \leq z_1$, $s > 1$ and $k \geq 1$, we have

$$k \cdot R - z_1(k - 1/s) \leq R$$

applying this to (18) yields

$$if^s_{(v,\bar{\pi})}(R) \leq R$$

2) $x < \frac{k \cdot s - s}{k \cdot s - 1}$. Since

$$\begin{aligned} & s \geq 1 + \frac{\sqrt{k^2 - k}}{k} \\ \Rightarrow \; & k \cdot s^2 - 2k \cdot s + 1 \geq 0 \\ \Rightarrow \; & s \geq 1 + \frac{k \cdot s - s}{k \cdot s - 1} \end{aligned}$$

combining this with $x < \frac{k \cdot s - s}{k \cdot s - 1}$ yields

$$\begin{aligned} & s \geq 1 + x \\ \Rightarrow & R \geq (R + if_{(v,\bar{\pi})}(z_1))/s \end{aligned}$$

and by (15) we know $R \geq if_{(v,\bar{\pi})}(z_2)/s$. On the other hand, by the definition of z_2 we know that z_2 is on some horizontal segment of $if_{(v,\bar{\pi})}(t)$, so by Lemma 3 we know $if^s_{(v,\bar{\pi})}(z_2) = if_{(v,\bar{\pi})}(z_2)/s$. In summary we have $if^s_{(v,\bar{\pi})}(z_2) \leq R$, and since $R < z_2$ we have

$$if^s_{(v,\bar{\pi})}(R) \leq R$$

In summary, in both cases it holds $if^s_{(v,\bar{\pi})}(R) \leq R$.

Since $\bar{\pi}$ is arbitrarily chosen from all path combinations, we have $\max_{\bar{\pi} \in \Pi(\tau)} \{ if^s_{(v,\bar{\pi})}(R) \} \leq R$. By Lemma 1 we have $ibf^s_{(v,\tau)}(R) \leq R$. Thus, the response time estimation returned by the IBF analysis on a speed-s processor is bounded by R. ■

When $k = 1$, $1 + \frac{\sqrt{k^2-k}}{k} = 1$, so when the interfering task set only has one task the speedup factor of IBF is 1, which gives the following corollary:

Corollary 1. IBF *is an exact analysis for dual-task systems.*

In general, the speedup factor $1 + \frac{\sqrt{k^2-k}}{k}$ of IBF is an increasing function with respect to k, as shown in Figure 10. That means the pessimism of IBF increases for task systems with more tasks This is because the *total* interference (bound) functions can increase with a larger slope with a larger k. The value of $1 + \frac{\sqrt{k^2-k}}{k}$ converges to 2 as k approaches infinity, which is the same as the speedup factor RBF, which corresponds to the fact that the slanted segments in $ibf_{(v,\tau)}(t)$ becomes "vertical" when infinitely many tasks release workload at the same time. There are also cases in which the maximal slope of $ibf_{(v,\tau)}(t)$ is smaller than the number of interfering tasks (when the interference bound function $ibf_T(t)$ of individual tasks does not increase at the same time). By a closer look into the proof of Theorem 7, we can see that the pessimism of the IBF analysis is actually decided by the maximal slope of $ibf_{(v,\tau)}(t)$. The IBF analysis guarantees to give exact analysis results if the Lipschitz constant of $ibf_{(v,\tau)}(t)$ is 1, regardless of the number of tasks in the system.

V. Evaluations

We use randomly generated task sets to evaluate the two approximate analysis methods RBF and IBF proposed in this paper, and compare them with the exact analysis in [18], denoted by EXACT. We define the utilization of a task T as the highest ratio of the sum of WCET vertex labels over the sum of edge labels in all cycles in $G(T)$. The total utilization of a task set is the sum of individual task utilizations.

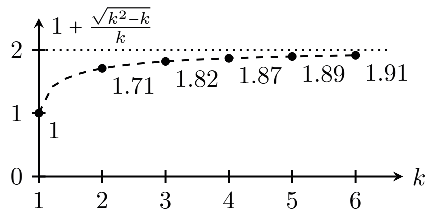


Figure 10. The values of $1+\frac{\sqrt{k^2-k}}{k}$ for different k.

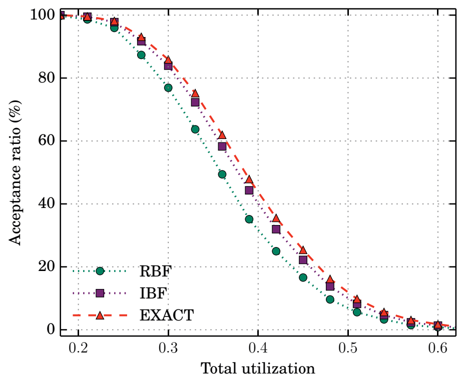


Figure 11. Acceptance ratio with different total utilizations.

To create a task, a random number (in the range $[7, 15]$) of vertices are generated. The output degree of vertex is randomly chosen in the range $[1, 6]$. Edges are randomly placed but it is guaranteed that the whole graph is strongly connected. We define different types of vertices, and the type of each vertex is randomly chosen. For each type, we specify the range of WCET values and the range of the release separations of any output edge of this vertex. For each vertex v, $e(v)$ and $p(v, u)$ for all edges (v, u) outgoing from v are randomly chosen from the corresponding range, and $d(v)$ is set to be the minimal $p(v, u)$ among all outgoing edges (v, u).

The procedure of generating task sets is as follows. First a task set of two tasks is constructed and evaluated. Then we randomly generate a new task and add it to the task set in last step, and repeat this procedure until the total utilization of the task set exceeds 1. Then a new task set of two newly generated tasks is constructed. The whole procedure repeats until a sufficiently large number of task sets are generated and evaluated.

Figure 11 shows the acceptance ratios of EXACT, RBF and IBF in different total utilization ranges. The x-axis represents the total utilization and the y-axis represents the *acceptance ratio*: Each point in the figure represents the ratio between the number of task sets decided as schedulable by a certain method and the total number of generated task sets, at a certain level of task set total utilization specified by its x-axis value. The experiments use two types of vertices. Two types of vertices are used for the experiments in Figure 11:

Type-I: $e(v) \in [1, 6]$ and $p(v, u) \in [20, 100]$.
Type-II: $e(v) \in [7, 9]$ and $p(v, u) \in [101, 300]$.

Figure 12 shows the acceptance ratios with different numbers of tasks in each task set. For this group of experiments we only use one vertex type: $p(v, u) \in [50, 300]$ and $e(v) \in [1, x]$ where x is tuned to control the range of utilization of each task, and thus control the number of tasks contained in each task set. For each target numbers of tasks, we select generated task sets with total utilizations in a certain range. The goal is to maintain the acceptance ratio of EXACT at a relatively stable level around 80%. The execution time upper bound x values to generate task sets with different target number of tasks are shown in Table I. From Figure 12 we can see that the superiority of IBF over RBF is larger for task sets with fewer tasks. In particular, IBF is as precise as EXACT for dual-task systems. These all coincide with the theoretical evaluation results using the speedup factor metric in Section IV.

We also compare the analysis efficiency of RBF, IBF and EXACT. The experiments use an implementation in Python and execute on a desktop computer with an Intel Core i7-2600 CPU (3.40GH). Experiments show that the analysis time of RBF and IBF are very close, and are much shorter than EXACT. In all of our experiments, the time consumption of RBF and IBF is at most 4 seconds, and is typically below 1 second. However, the time consumption of EXACT is very unstable. For some task sets, it is also quite efficient (almost as efficient as RBF and IBF), but for many task sets it is thousands of times slower than RBF and IBF. There are also a considerable portion of task sets that EXACT cannot finish the analysis in hours. In summary, the efficiency improvement of RBF and IBF against EXACT is significant.

number of tasks	2	3	4	5	6	7	8
x	33	20	15	12	10	8	7

Table I
DIFFERENT UPPER BOUNDS OF $e(v)$ TO GENERATE TASK SETS WITH DIFFERENT NUMBER OF TASKS.

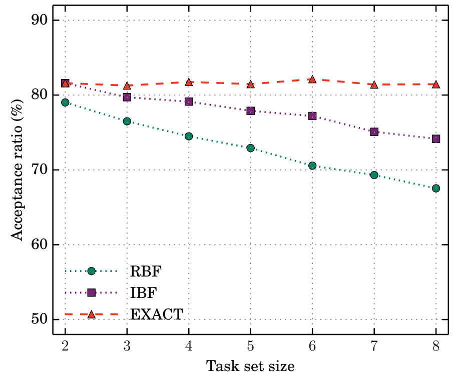


Figure 12. Acceptance ratio with different number of tasks.

VI. Conclusions and Future Work

In this paper we propose two approximate response time analysis methods for real-time task graph models, and quantify their suboptimality using the metric speedup factor. The first method RBF has a speedup factor of 2, which is tight even for task sets containing only two tasks. The second method

IBF has a speedup factor of $1 + \frac{\sqrt{k^2-k}}{k}$, where k is the number of interfering tasks with higher priority than the one under analysis. For the special case $k = 1$ this speedup factor equals 1, which implies that IBF gives exact analysis results for task sets containing two tasks. At the other extreme, IBF's speedup factor converges to 2 when k approaches to infinity. We also conduct experiments to empirically evaluate the precision and efficiency of RBF and IBF with randomly generated task sets. Results show that our proposed approximate analysis methods can achieve very high efficiency with low precision loss, and also confirm the trends indicated by the theoretical results of their speedup factors. In this work we assume the jobs all have constrained deadlines. In the future we will extend the analysis methods as well as speedup factor guarantees in this paper to allow deadlines larger than the inter-release separations on the outgoing edges.

VII. Acknowledgement

This work is partially supported by NSF of China (Grant No. of 61300022, 61370076 and 61472072).

References

[1] S. K. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS)*, 1990.

[2] Sanjoy Baruah. Feasibility analysis of recurring branching tasks. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 138–145. IEEE, 1998.

[3] Sanjoy Baruah. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[4] Sanjoy Baruah. The non-cyclic recurring real-time task model. In *Proceedings of the IEEE 31st Real-Time Systems Symposium (RTSS)*, pages 173–182, Nov 2010.

[5] Sanjoy K. Baruah, Deji Chen, Sergey Gorinsky, and Aloysius K. Mok:. Generalized multiframe tasks. *Real-Time Systems*, 1999.

[6] R. I. Davis, T. Feld, V. Pollex, and F. Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[7] J. Palencia Gutierrez and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, 1998.

[8] B. Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. In *Journal of ACM*, 2000.

[9] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. In *Journal of the ACM*, 1973.

[10] Jukka Maki-Turja and Mikael Nolin. Fast and tight response-times for tasks with offsets. In *Proceedings. 17th Euromicro Conference on Real-Time Systems (ECRTS)*.

[11] A. K. Mok and D. Chen. A multiframe model for real-time tasks. In *IEEE Transactions on Software Engineering*, 1997.

[12] Michealo Searcoid. Metric spaces. *Springer undergraduate mathematics series, Springer-Verlag, section 9.4*, 2006.

[13] Martin Stigge. Real-time workload models: Expressiveness vs. analysis efficiency. In *Ph.D. Dissertation, Uppsala University*, 2014.

[14] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The Digraph Real-Time Task Model. In *RTAS 2011*.

[15] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. On the tractability of digraph-based task models. *24th Euromicro Conference on Real-Time Systems (ECRTS)*, 2011.

[16] Martin Stigge, Nan Guan, and Wang Yi. Refinement-based exact response-time analysis. In *the 26th EUROMICRO Conference on Real-Time Systems (ECRTS)*, 2014.

[17] Martin Stigge and Wang Yi. Hardness results for static priority real-time scheduling. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 189–198. IEEE, 2012.

[18] Martin Stigge and Wang Yi. Combinatorial abstraction refinement for feasibility analysis. In *Procedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*, pages 340–349, 2013.

[19] H. Takada and K. Sakamura. Schedulability of generalized multiframe task sets under static priority assignment. In *Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications (RTCSA)*, 1997.

[20] Ken Tindell. Using offset information to analyse static priority preemptively scheduled task sets. In *Technical Report YCS-182, Dept. of Computer Science, University of York, England*, 1992.

Appendix: Proof of Lemma 2

We first introduce an auxiliary lemma to bound the total workload released in a "busy period" by the worst-case response time of the analyzed vertex.

Lemma 5. *Given a vertex v and an interfering task set τ. Suppose $[t_1, t_2)$ is some time interval during which the processor is continuously busy (executing jobs released by τ or v). The total workload of jobs released in $[t_1, t_2)$ is bounded by $R(v, \tau)$.*

Proof: We prove by contradiction, assuming the total workload of jobs released in $[t_1, t_2)$, denoted by W, is greater than R. Without loss of generality, we consider the case that v releases a job at t_1. Since the processor is continuously busy during $[t_1, t_2)$, the workload of jobs released in $[t_1, t_2)$ is finished no earlier than $t_1 + W$, i.e., the job released by v is finished no earlier than $t_1 + W$, which contradicts that R is v's worst-case response time. ■

Now we are ready to prove Lemma 2:

Proof: We prove the lemma for the case of $f(t) = if_{(v,\bar{\pi})}(t)$, and the case of $f(t) = rf_{(v,\bar{\pi})}(t)$ can be proved similarly.

Suppose each path $\pi_i \in \bar{\pi}$ releases its first job at time 0. Since z_1 is a lifting point of $if_{(v,\bar{\pi})}(t)$, we know that $if_{(v,\bar{\pi})}(z_1)$ is the total workload released before in time interval $[0, z_1)$, and since $if_{(v,\bar{\pi})}(z_1) \le z_1$, we know all the jobs released in $[0, z_1)$ have been finished by z_1, i.e., the processor is idle just before z_1. On the other hand, some job is released at z_1 since z_1 is a lifting point. So we can conclude that the processor becomes busy from idle at time z_1.

Then we prove that the processor is continuously busy during $[z_1, z_2)$. We prove this by contradiction, assuming that there is some point $z' \in (z_1, z_2)$ at which the processor is idle. Since z' is idle, all the workload released by z' has been finished by z', so z' must be in some horizontal segment of $if_{(v,\bar{\pi})}(t)$, and $if_{(v,\bar{\pi})}(z') \le z'$. Let z'' be the lifting point at the end of horizontal segment containing z', so

$$if_{(v,\bar{\pi})}(z'') \le if_{(v,\bar{\pi})}(z') = z' \le z''$$

so z_2 is a lifting point larger than z_1 satisfying $if_{(v,\bar{\pi})}(t) \le t$, which contradicts the definition of z_1. So we can conclude that the processor is continuously busy during $[z_1, z_2)$.

By Lemma 5 we know that the total workload of jobs released during $[z_1, z_2)$ is bounded by $R(v, \tau)$. On the other hand, since z_1 is a lifting point, we know $f(z_2) - f(z_1)$ only includes workload of jobs released during $[z_1, z_2)$. So putting the above discussions together, we have $f(z_2) - f(z_1) \le R(v, \tau)$. ■