Delay-Impact-Based Local Deadline Assignment for Online Scheduling of Distributed Soft Real-Time Applications

Miao Song†, Shuhui Li†, Shangping Ren†
Illinois Institute of Technology
Email: {msong8, sli38, ren}@iit.edu

Gang Quan‡
Florida International University
Email: {gang.quan}@fiu.edu

***Abstract*—Distributed soft real-time applications often involve multiple jobs that are executed on different processing units. Hence, resource competitions among these applications can be on any processing unit in the system. However, due to distributed nature of these applications, each processing unit may not have the knowledge about the workload on other processing units. Therefore, scheduling decisions made by individual processing units about their local job execution orders may not be optimal for the applications to which the jobs belong with respect to meeting the applications' end-to-end deadlines. In this paper, we first introduce a metric to measure, at a local processing unit, the risk of a distributed soft real-time application missing its end-to-end deadline. Second, based on the metric, we develop a local deadline assignment algorithm, i.e., the *delay-impact-based* (DIB) local deadline assignment algorithm. With the DIB algorithm, distributed processing units can independently schedule their local job sets based on the assigned job deadlines with maximized successful ratio of meeting distributed real-time applications' end-to-end deadlines. We empirically compare the DIB algorithm with three commonly used local deadline assignment algorithms, i.e., the OLDA, Pure, and Norm algorithms. The experimental results show that the DIB algorithm has clear advantage over the OLDA, Pure, and Norm approaches — it results in up to 50%, 35%, and 35% higher successful ratio than the OLDA, Pure, and Norm approaches with respect to meeting application's end-to-end deadlines, respectively. Furthermore, for those applications that do miss their end-to-end deadlines, the application execution delay ratio resulted by the DIB algorithm is also up to 300%, 50%, and 150% smaller comparing to the other three approaches.**

I. Introduction

Many distributed soft real-time applications, such as interactive online video gaming [1] and teleconferencing [2], involve multiple jobs executed on different processing units with specified end-to-end deadline requirements. These applications may be deployed to a system at any time and on any processing unit during run-time. For instance, a new computation intensive application may be deployed on the sparc workstation while an online gaming is in process. As a result, resource competitions among these applications can be on any processing unit and unpredictable. Furthermore, due to the distributed nature of the system, often times an individual processing unit does not have detailed information about the working status of other processing units. Hence, the schedule of applications on a processing unit has to be made at run-time and based on local information.

However, the resulting online schedule on a local processing unit is not necessarily optimal to ensure that all applications finish executions before their end-to-end deadlines. Even though the job assignment to particular processing units is known a priori, to determine the local scheduling order of applications in order to meet their end-to-end deadlines is still a NP-complete problem [3].

On each processing unit, the EDF scheduling algorithm is usually used to fully exploit processing resource, since EDF scheduling algorithm is an optimal scheduling algorithm for a single processing unit [4]. Accordingly, a local deadline should be assigned to a job when its application is dispatched to a processing unit. The jobs (their applications) are scheduled based on their local deadlines. We assume that all applications (their related jobs) continue their execution even if they pass their local deadlines or end-to-end deadlines for the following reasons: 1) missing a local deadline does not necessarily indicate the application will miss its end-to-end deadline, as the application may still have a chance to catch up during its future execution; 2) for soft real-time applications, an application missing its end-to-end deadline may still have its benefits [1].

In this paper, we introduce a metric to measure, at a local processing unit, the end-to-end deadline missing risk of each distributed soft real-time application. Based on the metric, we then develop a local deadline assignment algorithm, i.e., *delay-impact-based* (DIB) local deadline assignment algorithm to minimize the risk. With the job deadlines assigned by the DIB algorithm, distributed processing units can independently schedule their local job sets by the EDF scheduling algorithm to maximize successful ratio of meeting distributed real-time applications end-to-end deadlines.

We compare and analyze the proposed approach with three commonly used local deadline assignment approaches, i.e., the OLDA [2], Pure [5], and Norm [6], in terms of application successful ratio and application execution delay ratio (the degree of an application's end-to-end delay with respect to its deadline). Our experimental results show that our approach

† The research is supported in part by NSF under grant number CAREER 0746643, CNS 1018731 and CNS 1035894.

‡ The research is supported in part by NSF under grant number CNS 1423137 and CNS 1018108.

978-1-4799-7575-4/14/$31.00 ©2014 IEEE

presents significant advantages over the counterparts — it results in up to 50%, 35%, and 35% higher successful ratio than the OLDA, Pure, and Norm approaches with respect to meeting application's end-to-end deadlines, respectively. Furthermore, for those applications that do miss their end-to-end deadlines, the application execution delay ratio resulted by the DIB algorithm is also up to 300%, 50%, and 150% smaller comparing to the other three approaches.

The rest of the paper is organized as follows. We discuss related work in Section II. In Section III, we formulate the research problem the paper addresses. Our local deadline assignment algorithm is introduced in Section IV. Experimental results are discussed in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

For distributed real-time applications, how to guarantee the applications' end-to-end deadlines has been a research challenge and significant amount of efforts have been made in the research community. For instance, Bettati et.al. [7] developed a priority assignment approach that guarantees schedulability for the flow shop task model where each task's total execution length and execution time on the shared processing unit are identical. Abdelzaher et al. [8], [9] proposed a worst case bound for task graphs with/without cycles under the pipeline scheduling model. Based on the bound, they transformed the end-to-end scheduling problem into a uni-processor scheduling problem. Lee et al. [10] took the approach of allowing global viewpoint to be incorporated and developed a convex unified framework to guarantee end-to-end delay. These approaches either make special assumptions about the application models or assume that the global view is available to local processors in order to perform offline schedulability analysis.

Without end-to-end deadlines strictly guaranteed and strong coupling among scheduling decisions taken on each processing unit, another direction is to divide the end-to-end deadlines into local deadlines for each application's jobs on their execution processors and apply uni-processor scheduling algorithms to schedule local jobs. Kao et.al [5] proposed *Equal Slack* (Pure) and *Equal Flexibility* (Norm) local deadline assignment approaches for meeting end-to-end deadlines. Later research works exploited these two approaches and apply them into parallel task models. For instance, Natale et al. [3] gave a static slicing technique using a critical path that maximizes the minimum laxity. An adaptive deadline assignment technique that reflects the characteristics of task graph parallelism was presented in [6]. Although these approaches are straightforward and easy to implement, they share the same shortcoming: resource contention among jobs (applications) on different processing units is not taken into consideration when local deadlines are assigned.

In order to overcome the resource contention problem of distributed real-time applications, Hong et al. [2] recently proposed a laxity time based local deadline assignment approach. The objective of this approach is to maximize the minimal available laxity time of all jobs on each processing unit. However, considering only laxity time does not fully capture the nature of the risk of missing the end-to-end deadline due to the delay caused by resource contention. The performance of the approach can deteriorate sharply when all applications have to be finished even if they miss their end-to-end deadlines. In this paper, rather than using available slack time to decide an application's execution schedule, we study the impact of an application being delayed toward its end-to-end deadline and use the delay impact to decide job execution orders.

III. Problem Formulation

In this section, we first introduce models and definitions, and then formulate the research problem.

A. Models and Notations

System Model

The system consists of m autonomous processing units denoted as $\mathcal{V} = \{V_1, V_2, \cdots, V_m\}$. By autonomous, we mean each unit is independent, has its own job scheduler, and it does not have information about the operation status of other units. The system also contains a set of distributed soft real-time applications $\mathcal{A} = \{A_1, A_2, \cdots, A_n\}$ deployed on $\mathcal{V}$.

Application Model[1]

Each application has a set of precedence-related jobs which go through a chain of processing units. More precisely, a real-time application A_i is represented by a quadruple, i.e., $A_i \equiv (R_i, D_i, \Gamma_i, f_i)$, where

- R_i: application A_i's release time, we omit it if A_i is released at time 0.
- D_i: absolute end-to-end deadline;
- Γ_i: ordered execution sequence of jobs belonging to A_i. $\Gamma_i \equiv (J_{i,1}, J_{i,2}, \cdots, J_{i,l_i})$, where l_i is total number of jobs that application A_i has;
- $J_{i,k}$: the kth job in the ordered sequence of A_i. It is defined by $J_{i,k} \equiv (r_{i,k}, e_{i,k}, d_{i,k})$, $\forall k \in \{1, 2, \cdots, l_i\}$,
  - $r_{i,k}$: release time of $J_{i,k}$. Other than the first job's release time, i.e., $r_{i,1} = R_i$, all other jobs' release time $r_{i,k} (\forall k, k \neq 1)$ is decided at run-time;
  - $e_{i,k}$: execution time of $J_{i,k}$;
  - $d_{i,k}$: deadline of $J_{i,k}$, $d_{i,k} \leq D_i$. It does not have to be smaller than D_i and is also decided at run-time;
- f_i: a mapping between Γ_i and $\mathcal{V}$. $f_i(J_{i,k}) = V_p, V_p \in \mathcal{V}$, job $J_{i,k}$ is executed on V_p.

Notations and Definitions

$\boldsymbol{\Omega(V_p, t)}$: job set on processing unit V_p at time t.

***Exe*$\boldsymbol{(J_{i,k}, t)}$:** job $J_{i,k}$'s total accrued execution time since its release ($r_{i,k}$) to the current time t, $0 \leq Exe(J_{i,k}, t) \leq e_{i,k}$.

***Dly*$\boldsymbol{(J_{i,k}, t)}$:** to-be-delayed time of job $J_{i,k}$ at time t. $0 < Dly(J_{i,k}, t) \leq B$, where $B = \sum_{J_{l,m} \in \{\Omega(V_p,t) - \{J_{i,k}\}\}} (e_{l,m} - Exe(J_{l,m}, t))$. B implies the worst case to-be-delayed time of $J_{i,k}$: $J_{i,k}$ will be the last one executed among $\Omega(V_p, t)$.

[1] we do not have any constraints on the execution path of distributed soft real-time applications nor the execution time of their jobs on each deployed processing unit.

$\boldsymbol{Exe_{\text{app}}(A_i,t)}$: application A_i's total accrued execution time since its release (R_i) to the current time t. $Exe_{\text{app}}(A_i,t) = \sum_{k=1}^{l_i} Exe(J_{i,k},t) = \sum_{k=1}^{b-1} e_{i,k} + Exe(J_{i,b},t)$, assuming $J_{i,b}$ is currently being executed or waiting for being completed at time t and $J_{i,k}$ $(1 \le k \le b-1)$ has completed before or at time t.

$\boldsymbol{\mathcal{L}(A_i,t)}$: application A_i's laxity time at time t,

$$\mathcal{L}(A_i,t) = D_i - (\sum_{k=1}^{l_i} e_{i,k} - Exe_{\text{app}}(A_i,t)) - t \quad (1)$$

It indicates how much time A_i can be delayed without missing its end-to-end deadline at current time t.

$\boldsymbol{\eta(A_i)}$: application A_i's execution delay ratio. Assume A_i finishes its execution at time F_i,

$$\eta(A_i) = \begin{cases} \frac{F_i - D_i}{D_i} & \text{if } F_i > D_i \\ 0 & \text{if } F_i \le D_i \end{cases}$$

$\boldsymbol{\mathcal{A}_S}$: successful application set, i.e., the set of applications that finish their executions no later than the end-to-end deadlines, $\mathcal{A}_S \subseteq \mathcal{A}$.

$\boldsymbol{\gamma}$: application successful ratio, $\gamma = \frac{|\mathcal{A}_S|}{|\mathcal{A}|}$.

B. Preliminary

When distributed applications share processing units during their end-to-end executions, their jobs are not always immediately executed once released to a processing unit: some jobs need to wait for other jobs to finish. The finish time of a job (its corresponding application) on a processing unit depends on its execution order among the to-be-scheduled job set. If each job of an application is always the first one to execute on its deployed processing unit, then no laxity time is consumed and the application will not miss its end-to-end deadline. However, if a job is lastly executed, then it will need more than its own execution time to finish. Therefore, the related application has a higher risk of missing its end-to-end deadline in future execution. We call the risk of missing an application's end-to-end deadline due to waiting for other applications to finish "the delay impact".

How to measure the delay impact of each application and the scheduling decisions can reflect and minimize this impact when its related jobs are scheduled on local processing units is a challenge. A direct way [8] is to quantify the "delay impact" of an application as its available laxity time after being delayed on the shared processing unit. The smaller available laxity time an application has, the higher risk the application misses its end-to-end deadline. The application with the largest available laxity time among a scheduled application set is executed last to minimize the risk of the whole set. We use the *Example 1* to demonstrate how it works:

Example 1: Assume that there are three processing units in the system $\{V_1, V_2, V_3\}$ and three applications A_1, A_2 and A_3 as denoted in Fig 1.

Assume all three applications have arrival time of 0[2] and thus their release time are omitted in the following formula.

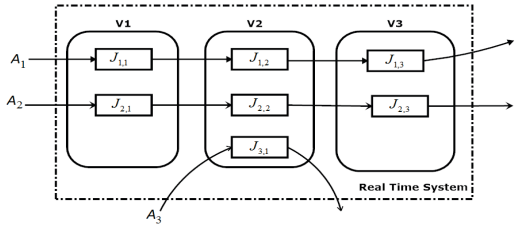


Fig. 1. Application Execution Paths

$$A_1 = (77, ((0,24,?),(?,27,?),(?,15,?)), \{f_1(J_{1,1}) = V_1, f_1(J_{1,2}) = V_2, f_1(J_{1,3}) = V_3\})$$
$$A_2 = (78, ((0,9,?),(?,23,?),(?,9,?)), \{f_2(J_{2,1}) = V_1, f_2(J_{2,2}) = V_2, f_2(J_{2,3}) = V_3\})$$
$$A_3 = (100, ((0,37,?)), \{f_3(J_{3,1}) = V_2\})$$

where "?" denotes information that needs to be decided at run-time.

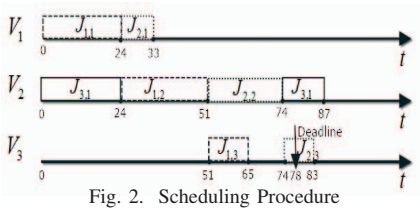


Fig. 2. Scheduling Procedure

1) At $t = 0$, assume two jobs $J_{1,1}(A_1)$ and $J_{2,1}(A_2)$ are released to V_1 and $J_{3,1}(A_3)$ is released to V_2 . Since $J_{3,1}$ has no other jobs to compete with on V_2, $J_{3,1}$ will be immediately executed on V_2. Hence, the local deadline for $J_{3,1}$ is set as 37 ($d_{3,1} = 37$). On the other hand, $J_{1,1}$ and $J_{2,1}$ have to be scheduled based on a certain order. If $J_{1,1}$ is the last one to execute, its finish time will be 33. By time 33, $\mathcal{L}(A_1,33) = D_1 - (\sum_{k=1}^{3} e_{1,k} - Exe_{\text{app}}(A_1,33)) - 33 = 77 - (66-24) - 33 = 2$. If $J_{2,1}$ is the last one to execute, its finish time will be 33. $\mathcal{L}(A_2,33) = D_2 - (\sum_{k=1}^{3} e_{2,k} - Exe_{\text{app}}(A_2,33)) - 33 = 78 - (41-9) - 33 = 13$. Since $\mathcal{L}(A_2,33) > \mathcal{L}(A_1,33)$, we set $J_{1,1}$ to be the first one and $J_{2,1}$ to be the last one to execute on V_1. As a result, the local deadline for $J_{1,1}$ is set as 24 ($d_{1,1} = 24$), the local deadline of $J_{2,1}$ is set as 33 ($d_{2,1} = 33$).

2) At $t = 24$, $J_{1,1}$ finishes execution on V_1 and $J_{1,2}$ is thus dispatched on V_2. Hence, $\Omega(V_2,24) = \{J_{1,2}, J_{3,1}\}$. $Exe(J_{3,1},24) = 24$ and $J_{3,1}$ has 13 time units left to be executed. If we execute $J_{1,2}$ last on V_2, then $J_{1,2}$ finishes execution at time 64, $\mathcal{L}(A_1,64) = -2$. If $J_{3,1}$ is executed last instead, $\mathcal{L}(A_3,64) = 36$. $J_{1,2}$ is thus executed first, after which $J_{3,1}$ is executed. Therefore, $d_{1,2} = 51, d_{3,1} = 64$.

3) At $t = 33$, $J_{2,1}$ finishes execution on V_1 and $J_{2,2}$ is dispatched on V_2. $\Omega(V_2,33) = \{J_{1,2}, J_{2,2}, J_{3,1}\}$. $Exe(J_{1,2},33) = 9$ and $J_{1,2}$ has 18 time units left to execute. $Exe(J_{2,2},33) = 0$ and $J_{2,2}$ has 23 time units left. $Exe(J_{3,1},33) = 24$ and $J_{3,1}$ has 13 time units left to execute. If $J_{1,2}$ is the last one to execute, then $\mathcal{L}(A_1,87) = -25$. Similarly, $\mathcal{L}(A_2,87) = -18$, $\mathcal{L}(A_3,87) = 13$. The order of execution is $J_{1,2}, J_{2,2}$ and $J_{3,1}$, and $d_{1,2} = 51, d_{2,2} = 74, d_{3,1} = 87$.

[2]This paper does not assume that all applications release at time 0 or at the same time point. The special release assumption in this example serves for the convenience of explanation.

4) At $t = 51$, $J_{1,2}$ finishes execution on V_2 and $J_{1,3}$ is dispatched on V_3. $J_{2,2}$, $J_{3,1}$ continue their execution on V_2.
5) At $t = 74$, $J_{2,2}$ finishes execution on V_2 and $J_{2,3}$ is dispatched on V_3. $J_{1,3}$ finished executing on V_3 and left the system at $t = 65$. $J_{2,3}$ is immediately executed without competition with $J_{1,3}$ on V_3 and will finish its execution at 83. $J_{3,1}$ continues its execution on V_2 and will finish at 87.

In this example, A_1 and A_3 finish their execution before their end-to-end deadlines, but A_2 fails to meet its end-to-end deadline. Hence, the application successful ratio $\gamma = \frac{2}{3} = 67\%$ and the delay execution ratio for J_2 is $\eta(A_2) = \frac{83-78}{78} = 6\%$. The detailed scheduling procedure of jobs on each processing unit is shown in Fig. 2. □

However, in fact, all A_1, A_2 and A_3 can finish successfully before their end-to-end deadlines. In other words, available laxity time is not the best way to measure the delay impact of individual applications.

If an application is delayed, the possibility that it can still finish before its end-to-end deadline is affected by three aspects: 1) the available remaining laxity time; 2) the total remaining execution demand; and 3) the possible competition with other applications during its future execution. A local processing unit cannot predict the potential competition of the scheduled applications on downstream processing units; it can, however, have the knowledge of the remaining laxity time and remaining execution demand of each application. The delay impact of an application, which a processing unit should consider when deciding job execution order, must take into consideration of these two aspects.

For application A_i, assume its corresponding job $J_{i,k}$ is being executed or waiting for being completed at time t. If the to-be-delayed time of $J_{i,k}$ is $Dly(J_{i,k}, t)$ and the remaining execution of $J_{i,k}$ is $e_{i,k} - Exe(J_{i,k}, t)$, then $J_{i,k}$ will finish its execution at $f_{i,k} = t + Dly(J_{i,k}, t) + e_{i,k} - Exe(J_{i,k}, t)$. As shown in Fig. 3, the time interval between the current time t and its end-to-end deadline D_i consists of three parts: $Dly(J_{i,k}, t)$, remaining work of A_i, i.e., $\sum_{k^*=1}^{l_i} e_{i,k^*} - Exe_{\text{app}}(A_i, t)$ ($Exe_{\text{app}}(A_i, t) = Exe_{\text{app}}(A_i, t + Dly(J_{i,k}, t)) = Exe_{\text{app}}(A_i, f_{i,k}) - (e_{i,k} - Exe(J_{i,k}, t))$), and remaining laxity time $\mathcal{L}(A_i, f_{i,k})$.

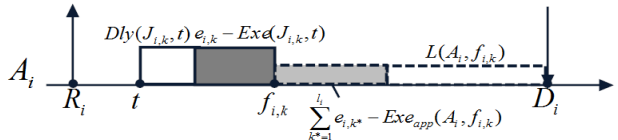


Fig. 3. Time Interval Between Current Time t and D_i

At the scheduling point t, depending on the scheduled position of $J_{i,k}$, it will have different $Dly(J_{i,k}, t)$. Accordingly, A_i will have different remaining work $\sum_{k^*=1}^{l_i} e_{i,k^*} - Exe_{\text{app}}(A_i, t)$ and $\mathcal{L}(A_i, f_{i,k})$ after it is delayed by $Dly(J_{i,k}, t)$. Rather than measuring the impact of $Dly(J_{i,k}, t)$ on A_i's end-to-end deadline missing risk by $\mathcal{L}(A_i, f_{i,k})$, a fair way is to measure the impact by the portion which delayed time $Dly(J_{i,k}, t)$ has accounted within A_i valid relative deadlines ($D_i - t$).

Based on the above information, we have the following definition:

$\alpha(A_i, t)$: the delay impact factor of A_i at time t,

$$\alpha(A_i, t) = \frac{Dly(J_{i,k}, t)}{\sum_{k^*=1}^{l_i} e_{i,k^*} - Exe_{\text{app}}(A_i, t) + \mathcal{L}(J_{i,k}, f_{i,k})} \quad (2)$$

where $f_{i,k} = t + Dly(J_{i,k}, t) + e_{i,k} - Exe(J_{i,k}, t)$.

Lemma 1: $\forall t, R_i \le t \le D_i$, if $\alpha(A_i, t) = 0$, then A_i is guaranteed to meet its end-to-end deadline.

Proof: the proof is trivial. Since A_i does not consume any laxity time waiting for the other jobs' execution at any time during its end-to-end execution, the finish time of A_i is its end-to-end execution time $\sum_{k=1}^{l_i} e_{i,k}$, which is not larger than D_i. Therefore, A_i is guaranteed to meet its end-to-end deadline. □

It is clear that the smaller $\alpha(A_i, t)$ is, the higher the possibility that A_i meets its end-to-end deadline. Our goal is to maximize the successful ratio for a given number of distributed soft real-time applications released in the system, i.e., minimize the delay impact factor of each application released in the system. At a local processing unit, since it is ambiguous to require minimizing the delay impact factor of all the scheduled applications, our objective becomes minimizing the maximum delay impact factor among all the applications whose jobs are executed on the shared local processing unit.

Application's delay impact factors are decided by their scheduled order, therefore a proper scheduled order of applications is a key to achieve the objective. As we schedule jobs on local processing units based on EDF scheduling algorithm, the scheduled order of the jobs is actually decided by the local deadlines assigned to each job.

Research Problem

Given $\mathcal{A}$ and $\mathcal{V}$ as defined before, let $\Omega(V_p, t)$ denote the job set to be scheduled on processing unit V_p at a scheduling point t where $V_p \in \mathcal{V}$. $\Omega(V_p, t)$ is scheduled based on the EDF scheduling algorithm. We have the following optimization problem: decide local deadline $d_{i,k}$ for each $J_{i,k}$ on processing unit V_p in order to:

$$\min : \max_{J_{i,k} \in \Omega(V_p, t)} \alpha(A_i, t) \quad (3)$$

$$s.t.\ r_{i,k} + e_{i,k} \le d_{i,k} \quad (4)$$

$$d_{i,k} - r_{l,m} \le \sum_{\substack{J_{h,n} \in \Omega(V_p, t), \\ r_{h,n} \ge r_{l,m}, \\ d_{h,n} \le d_{i,k}}} e_{h,n},\ where\ J_{l,m} \in \Omega(V_p, t) \quad (5)$$

Formula (4) indicates the lower bound of a valid $d_{i,k}$; formula (5) is the schedulability constraint for jobs in $\Omega(V_p, t)$ to be scheduled by EDF [11].

IV. Delay-Impact-Based Local Deadline Assignment

When multiple jobs are competing for shared resource, their delayed time depends on their scheduled order. For a single application, its delayed time gets larger when its job is scheduled later. Therefore, an application's delay impact

factor has the largest value if its job is the last one scheduled among all the possible scheduled positions. For a given job set $\Omega(V_p,t)$, we propose the following criterion to select a job as the last one to schedule.

Property 1: If J_{i^*,k^*} is the last job to schedule among $\Omega(V_p,t)$ at time t, then

$$\alpha(A_{i^*},t) \le \alpha(A_i,t), \forall J_{i,k} \in \Omega(V_p,t) \; where$$
$$Dly(J_{i,k},t) = \sum_{J_{l,m} \in \{\Omega(V_p,t)-\{J_{i,k}\}\}} (e_{l,m} - Exe(J_{l,m},t))$$

□

For $J_{i,k}$ ($\forall J_{i,k} \in \Omega(V_p,t)$), if it is the last one to be scheduled among $\Omega(V_p,t)$, then it has its longest to-be-delayed time among its possible scheduled positions: $Dly(J_{i,k},t) = \sum_{J_{l,m} \in \{\Omega(V_p,t)-\{J_{i,k}\}\}} (e_{l,m} - Exe(J_{l,m},t))$. By calculating each job in $\Omega(V_p,t)$ with its longest to-be-delayed time and its application's corresponding delay impact factor, *Property 1* chooses a job with the minimum delay impact factor to schedule lastly.

The last scheduled job chosen by *Property 1* has the latest finish time within $\Omega(V_p,t)$. We thus assign the latest finish time as its local deadline to guarantee its EDF-based scheduling order. We call this *Delay-Impact-Based local deadline assignment* (DIB), which is described in Algorithm 1.

Algorithm 1: DIB($\Omega(V_p,t)$)

```
1  while Ω(V_p,t) ≠ φ do
2      Max_Deadline = t + Σ_{J_{i,k}∈Ω(V_p,t)} (e_{i,k} − Exe(J_{i,k},t))
3      Min_Delay_Impact_Factor = ∞
4      foreach J_{i,k} ∈ Ω(V_p,t) do
5          Dly(J_{i,k},t) = Σ_{J_{l,m}∈{Ω(V_p,t)−{J_{i,k}}}} (e_{l,m} − Exe(J_{l,m},t))
6          if α(A_i,t) < Min_Delay_Impact_Factor then
7              Min_Delay_Impact_Factor = α(A_i,t)
8              i* = i
9          end
10     end
11     d_{i*,k*} = Max_Deadline
12     Ω(V_p,t) = Ω(V_p,t) − J_{i*,k*}
13 end
```

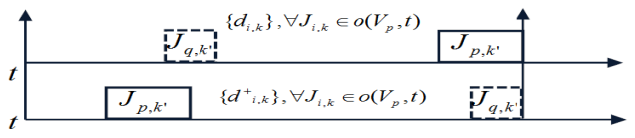


Fig. 4. The Execution Order Of Jobs In $o(V_p,t)$

In line 2, DIB calculates the maximum local deadline, i.e., the latest finish time of $\Omega(V_p,t)$, to be assigned to jobs. From line 4 to line 10, the algorithm chooses the job that satisfies *Property 1*. Line 11 assigns the maximum local deadline to the chosen job, then the job is removed from $\Omega(V_p,t)$ in line 12. The same procedure repeats until all the jobs in $\Omega(V_p,t)$ are assigned with respective local deadlines, i.e., they are assigned with proper scheduling orders. As the size of $\Omega(V_p,t)$ is bounded by $|\mathcal{A}| = n$, the time complexity of DIB is bounded by $O(n^2)$.

Theorem 1: Given the job set $\Omega(V_p,t)$, let $d_{i,k}$ be the local deadline assigned to each $J_{i,k} \in \Omega(V_p,t)$ by DIB, then $d_{i,k}$ minimizes the given objective in (3).

Proof: Suppose there exists an optimal set of local deadlines $\{d^+_{i,k}\}$ that is different from the solution $\{d_{i,k}\}$ obtained by DIB. Then, there exists at least one job $J_{i,k}$ whose $d^+_{i,k}$ is different from $d_{i,k}$. Let the jobs in $\Omega(V_p,t)$ be examined in the order of the sequence that a job obtains its local deadline by DIB, i.e., the first examined job $J_{1,k}$ has the largest local deadline among $\Omega(V_p,t)$, the second examined job has the largest local deadline among $(\Omega(V_p,t) - \{J_{1,k}\})$, and the last examined job $J_{n,k}$ ($n = |\Omega(V_p,t)|$) has its local deadline equal to $t + e_{n,k} - Exe(J_{n,k},t)$. Suppose $J_{x,k'}$ is the first job that has different deadlines $d^+_{x,k'}$ and $d_{x,k'}$ in solutions $\{d^+_{i,k}\}$ and $\{d_{i,k}\}$, respectively. The job set which excludes previously examined jobs is denoted as $o(V_p,t)$, $o(V_p,t) \subseteq \Omega(V_p,t)$. The execution orders of jobs in $o(V_p,t)$ decided by $\{d_{i,k}\}$ and $\{d^+_{i,k}\}$ respectively are shown in Fig. 4.

According to DIB, $J_{x,k'}$ has the largest local deadline among $o(V_p,t)$; assume in $\{d^+_{i,k}\}$, $J_{y,k'}$ has the largest local deadline among $o(V_p,t)$, $x \neq y$.

According to *Property 1*, $\alpha(A_x,t) \le \alpha^+(A_y,t)$. Also, $\alpha^+(A_x,t) < \alpha(A_x,t)$, as $Dly^+(J_{x,k'},t) < Dly(J_{x,k'},t)$. Therefore, $\alpha^+(A_x,t) < \alpha^+(A_y,t) \le \max_{J_{i,k} \in o(V_p,t)} \{\alpha^+(A_i,t)\}$. $\alpha^+(A_x,t)$ does not influence the objective function $\max_{J_{i,k} \in o(V_p,t)} \{\alpha^+(A_i,t)\}$ directly in solution $\{d^+_{i,k}\}$.

We have $\alpha(A_x,t) \le \alpha^+(A_y,t)$, in addition, $\alpha(A_y,t) < \alpha^+(A_y,t)$, as $Dly(J_{y,k'},t) < Dly^+(J_{y,k'},t)$. Therefore, the delay impact factor of A_x and A_y obtained by solution $\{d_{i,k}\}$ is smaller than or equal to that obtained by $\{d^+_{i,k}\}$. If $J_{x,j'}$ and $J_{y,j'}$ are removed from $o(V_p,t)$, the same proof also applies to the remaining jobs. As a result, the value of the objective function obtained by DIB is smaller or equal to the objective function of solution $\{d^+_{i,k}\}$. Hence, the solution found by DIB is optimal. □

Let us revisit *Example 1* to demonstrate how DIB is applied to the example.

1) At $t = 0$, $\Omega(V_1,0) = \{J_{1,1}, J_{2,1}\}$. If $J_{1,1}$ is the last one to execute on V_1, then $\alpha(A_1,0) = \frac{Dly(J_{1,1},0)}{\sum_{k=1}^{3} e_{1,k} - Exe_{\text{app}}(A_1,0) + \mathcal{L}(J_{1,1},33)} = \frac{9}{66-0+2} = 0.13$. If $J_{2,1}$ is the last one to execute instead, $\alpha(J_{2,1},0) = 0.45$. Since $\alpha(A_1,0) < \alpha(A_2,0)$, $J_{2,1}$ is assigned to be the first one to execute and $J_{1,1}$ is assigned to be the last one to execute. Therefore, $d_{2,1} = 9, d_{1,1} = 33$.

2) At $t = 9$, $J_{2,1}$ finishes execution on V_1 and $J_{2,2}$ is dispatched on V_2. $Exe(J_{3,1},9) = 9$ and $J_{3,1}$ has 28 times units left to complete execution. $\Omega(V_2,9) = \{J_{2,2}, J_{3,1}\}$. $\alpha(A_2,9) = 0.68$, $\alpha(A_3,9) = 0.34$. As $\alpha(A_3,9) < \alpha(A_2,9)$, then we have $d_{2,2} = 32, d_{3,1} = 60$. $J_{1,1}$ will continue its execution on V_1.

3) At $t = 32$, $J_{2,2}$ finishes execution on V_2 and $J_{2,3}$ is dispatched on V_3.

4) At $t = 33$, $J_{1,1}$ finishes execution on V_1, and $J_{1,2}$ is dispatched on V_2. $Exe(J_{3,1},33) = 10$ and $J_{3,1}$ has 27 time units left to complete execution. $\Omega(V_2,33) = \{J_{1,2}, J_{3,1}\}$. $\alpha(A_1,33) = 1.59$, $\alpha(A_3,33) = 0.675$. Since $\alpha(A_3,33) < \alpha(A_1,33)$, we have

$d_{1,2} = 60, d_{3,1} = 87$. $J_{2,3}$ will continue its execution on V_3 and finishes at 41.

As a result, A_1, A_2, and A_3 all successfully finish their execution at 75, 41, 87, respectively. The detailed scheduling procedure of jobs on each processing unit is shown in Fig. 5. □

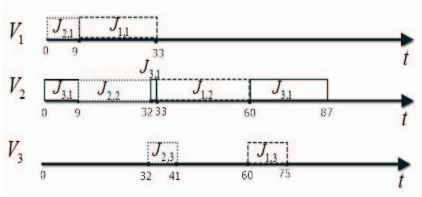


Fig. 5. Scheduling Procedure

V. Performance Evaluation

In this section, we empirically evaluate our DIB algorithm with three other commonly used local deadline assignment algorithms, i.e., the OLDA [2], Pure [5], and Norm [6]. The OLDA dynamically assigns local deadline that aims to maximize the laxity time to increase the application successful ratio. The Pure and Norm are two static local deadline assignment algorithms which assign total laxity time evenly or proportional onto each job, respectively. Our evaluation and comparison focus on two aspects, i.e., application successful ratio and application execution delay ratio.

Application Type and Instance Generation

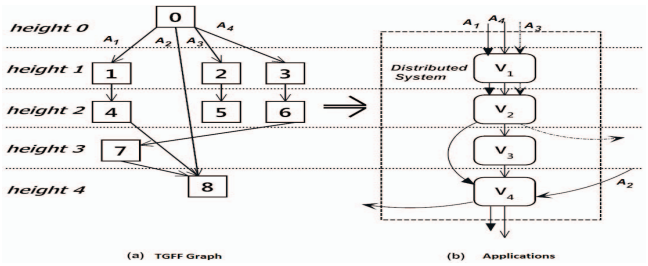


Fig. 6. Application Type Generation

TGFF [12] is a well known tool that generates various task graphs based on real applications. It is often used to generate benchmark applications in real-time community. Therefore, we use TGFF to randomly generate different types of distributed real-time applications. By application type, we mean an application structure, i.e., job sequences and their execution processing units. The nodes in a TGFF graph have precedence constraints, which correlates to the execution order of interdependent jobs belonging to an application. Based on the structure of an TGFF graph, we do the following mapping: 1) a node in a TGFF graph corresponds to a job; 2) a direct edge in a TGFF graph corresponds to a precedence constraint on the execution order between two jobs in the same application; 3) a path in a TGFF graph corresponds to an execution order of jobs in an application; 4) the number of paths corresponds to the number of applications in the system; 5) the height of a TGFF graph corresponds to the number of processing units in the system. Fig. 6(a) shows a task graph generated by TGFF. Based on the height from the root node (node 0) to the sink node (node 8), the graph has four layers. Hence, we assume there are four processing units. The jobs that are on the same layer are assigned to the same processing unit as shown in Fig. 6(b)[3].

Once the structure of the application is generated, the next step is to generate timing information for each application, i.e., the end-to-end execution time, the end-to-end deadline, and the execution time for each job within the application(workload on each processing unit). We therefore obtain concrete application instances. Depending on the execution time distribution among jobs within an application, we characterize two different workload scenarios: 1) balanced workload, i.e., the execution time of different jobs within an application has a high probability to be the same; and 2) unbalanced workload, i.e., the execution time of different jobs within an application has a high probability to be different. We use the methods presented in [13] to generate the two different types of workloads.

System Load Generation

As system load can be changed from many sources, we conduct the experiments from three aspects to monitor the performance of the four local deadline assignment approaches: 1) The applications/processors ratio (β). As the number of processing units in the system is fixed, by increasing β, the number of applications released to the system will increase, so does the system load. 2) The application execution density (ε): $\varepsilon = \sum_{k=1}^{l_i} e_{i,k}/D_i$. The increasing of ε indicates that the workload demand released to the system will also increase. 3) The end-to-end deadline distribution (δ). δ mimics the end-to-end deadline variance among application set from the average end-to-end deadline (D_{avg}). In particular, the end-to-end deadline of applications varies between $[D_{avg}(1-\delta), D_{avg}(1+\delta)]$. Increasing δ implies a decrease of the system workload.

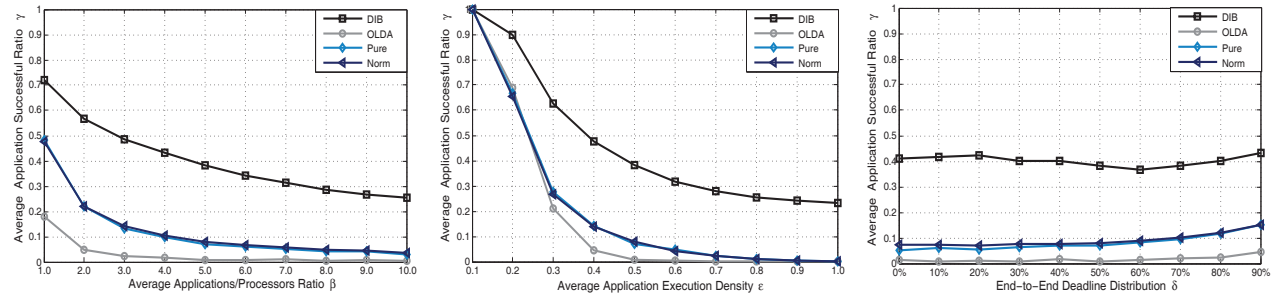
Experiment Setting

Applications and their execution paths are randomly generated by TGFF. For each application type generated by TGFF, a set of application instances is generated with concrete parameters for processing units to execute. The experiment is repeated on 5 different application types, i.e., 5 different TGFF graphs, with 500 different sets of application instances. We take the average value of the $5 \times 500 = 2500$ tests. We set the number of processing units in the system as 40 and the average end-to-end deadline value of jobs (D_{avg}) as 1000. We have run 10 groups of experiments under each case of system load generation, the mapping between group index and the detailed parameters are shown in the following table:

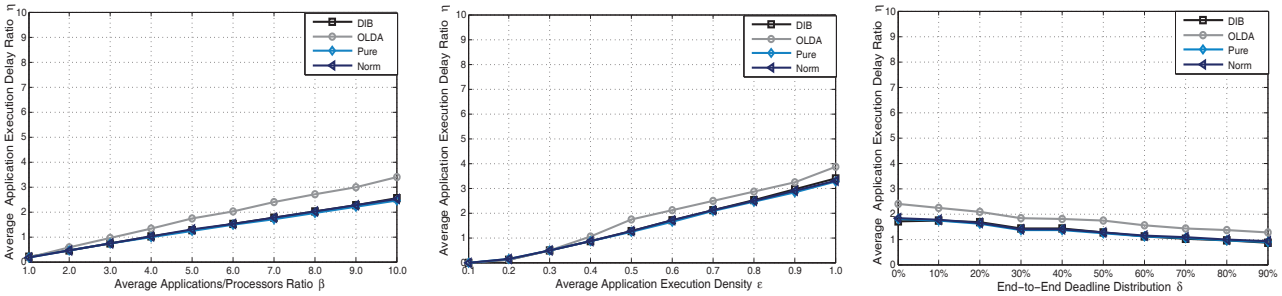
Group Index	β $\varepsilon = 0.5$ $\delta = 50\%$	ε $\beta = 5.0$ $\delta = 50\%$	δ $\beta = 5.0$ $\varepsilon = 0.5$
1	1.0	0.1	0%
...	...	...	...
10	10.0	1.0	90%

Experiment Results

[3]It is worth noting that Fig. 6 only serves as an example to show how we can transfer a TGFF graph to an application type. In the experiments, a set of TGFF graphs are randomly generated to have a broader coverage on different application types.

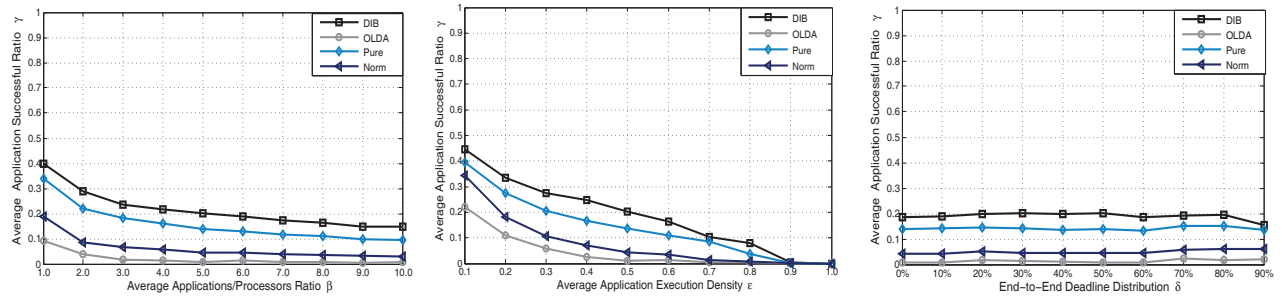


(a) $\varepsilon = 0.5, \delta = 50\%$ (b) $\beta = 5.0, \delta = 50\%$ (c) $\varepsilon = 0.5, \beta = 5.0$

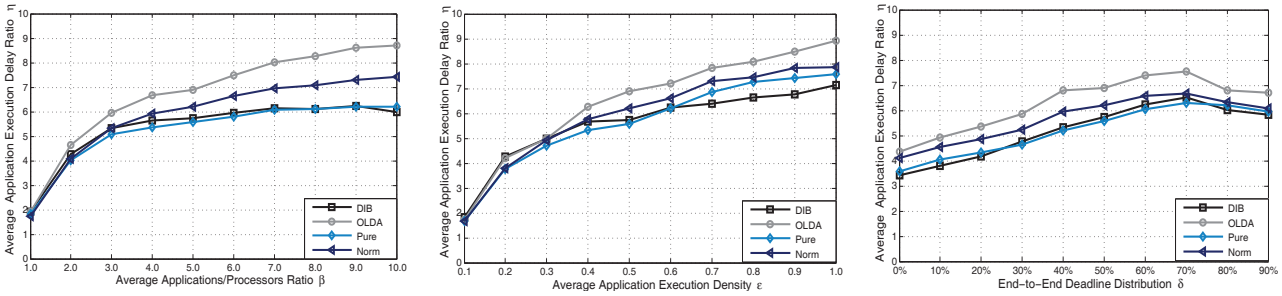


(d) $\varepsilon = 0.5, \delta = 50\%$ (e) $\beta = 5.0, \delta = 50\%$ (f) $\varepsilon = 0.5, \beta = 5.0$

Fig. 7. Balanced Workload



(a) $\varepsilon = 0.5, \delta = 50\%$ (b) $\beta = 5.0, \delta = 50\%$ (c) $\varepsilon = 0.5, \beta = 5.0$



(d) $\varepsilon = 0.5, \delta = 50\%$ (e) $\beta = 5.0, \delta = 50\%$ (f) $\varepsilon = 0.5, \beta = 5.0$

Fig. 8. Unbalanced Workload

Under a balanced workload, the average application successful ratio and average application execution delay ratio relating to the change of β, ε, and δ, respectively, are shown in Fig. 7. From Fig. 7, we can make the following observations:

1) The average successful ratio decreases and the average application execution delay ratio increases along with the increase of system load. The increasing of system load indicates that more applications will miss their end-to-end deadlines, which results in a smaller application successful ratio and a larger application execution delay ratio.

2) It is clear that the DIB performs the best among the four approaches with as much as 50%, 35%, 35% larger

average successful application ratio than the OLDA, Pure, and Norm, respectively. The DIB also results in as much as 100% smaller average application execution delay ratio than the OLDA algorithm. The OLDA performs the worst. This is because the OLDA dynamically assigns local deadlines to jobs of applications only based on available laxity time. It works better than the Pure and Norm when job removals are considered, since the latter two algorithms statically assign local deadlines without considering resource competition. If there is a heavy competition among applications on a limited resource, many jobs (applications) missing their assigned local deadlines will be removed. However, when removing jobs that miss their local deadlines is not considered, the OLDA cannot take advantage of obtaining a larger application successful ratio by removing a few jobs when they miss local deadlines. Instead, it encounters a larger application execution delay ratio and a smaller successful ratio. On the other hand, the Pure and Norm can catch up during execution at later stages even if the jobs miss the local deadlines at early stages. As the DIB considers delay impact at every step during scheduling process, its performance is the best.

3) Under a balanced workload, the performance of the Norm and Pure converges. This is because the deadline assignment results of the two approaches does not make much difference if the execution time of different jobs of an application has a high probability to be the same.

4) Since δ adjusts the variation of end-to-end deadlines of applications, it does not have the direct and significant contribution to the change of workload as β and ε do. As a result, the change of δ does not make a large impact either on applications' average successful ratio or on execution delay ratio comparing with β and ε. The variance of all four algorithms with respect to the average application successful ratio is within 5%, and within 100% with respect to the average application execution delay ratio.

Under an unbalanced workload, the average successful application ratio and average application execution delay ratio is shown in Fig. 8.

1) Comparing Fig. 8 with Fig. 7, the performance of the system decreases under an unbalanced workload with a smaller average application successful ratio and a bigger average application execution delay ratio. Take the DIB for example, the decrease of average application successful ratio and increase of average application execution delay ratio is as much as 55% and 350%, respectively. This is because when the execution time of different jobs in an application has a high probability to be different, the jobs of different applications with larger execution time may all compete on the same processing units. Under such scenarios, some processing units become the "bottleneck", while others have less burden. Therefore, the performance of the system deteriorates due to the unbalanced local processing unit utilization.

2) The DIB still performs the best among the four approaches, while the OLDA performs the worst. The difference between the DIB and OLDA is up to 25% with respect to average application successful ratio and 280% with respect to the average application execution delay ratio.

3) The performances of the Norm and Pure diverge: the Pure performs better than the Norm with up to 10% larger average application successful ratio and up to 150% smaller average application execution delay ratio.

VI. Conclusion

For distributed real-time applications, due to resource contention and unavailable global information at local processing units, deciding an execution order of jobs to meet their applications' end-to-end deadlines is a challenge. In this paper, we first present a "delay impact factor" to measure, at a local processing unit, the risk of missing an application's end-to-end deadline at run-time. Based on that, we then develop the delay-impact based (DIB) local deadline assignment algorithm to assign local deadlines to jobs of distributed soft real-time applications, so that the execution order of jobs according to such deadlines can maximize the application's successful ratio. We theoretically prove that DIB presents the local optimal solution to minimize the delay impact of the scheduled application set. Our experimental results also show significant advantages of the DIB over the counterparts in terms of larger application successful ratio and smaller application execution delay ratio.

References

[1] L. R. Welch and S. Brandt, "Toward a realization of the value of benefit in real-time systems," in *Parallel and Distributed Processing Symposium, International*, vol. 3. IEEE Computer Society, 2001, pp. 30 093a–30 093a.

[2] S. Hong, T. Chantem, and X. Hu, "Meeting end-to-end deadlines through distributed local deadline assignments," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE, 2011, pp. 183–192.

[3] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Real-Time Systems Symposium, 1994., Proceedings*. IEEE, 1994, pp. 216–227.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[5] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 12, pp. 1268–1274, 1997.

[6] J. Jonsson and K. G. Shin, "Robust adaptive metrics for deadline assignment in distributed hard real-time systems," *Real-Time Systems*, vol. 23, no. 3, pp. 239–271, 2002.

[7] R. Bettati, "End-to-end scheduling to meet deadlines in distributed systems," Ph.D. dissertation, University of Illinois, 1994.

[8] P. Jayachandran and T. Abdelzaher, "Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling," in *Real-Time Systems, ECRTS'08. Euromicro Conference on*. IEEE, 2008, pp. 233–242.

[9] P. J. and T. A., "End-to-end delay analysis of distributed systems with cycles in the task graph," in *Real-Time Systems, ECRTS'09. Euromicro Conference on*. IEEE, 2009, pp. 13–22.

[10] J. Lee, I. Shin, and A. Easwaran, "Convex optimization framework for intermediate deadline assignment in soft and hard real-time distributed systems," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2331–2339, 2012.

[11] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, vol. 2, no. 3, pp. 181–194, 1990.

[12] R. Dick, D. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*. IEEE Computer Society, 1998, pp. 97–101.

[13] E. Bini and G. Buttazzo, "Biasing effects in schedulability measures," in *Real-Time Systems, 2004. ECRTS'04. Proceedings. 16th Euromicro Conference on*. IEEE, 2004, pp. 196–203.