Meeting End-to-End Deadlines Through Distributed Local Deadline Assignments

Shengyan Hong, Thidapat Chantem, Xiaobo Sharon Hu
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
{shong3, tchantem, shu}@nd.edu

Abstract— **In a distributed real-time system, jobs are often executed on a number of processors and must be completed by their end-to-end deadlines. Without considering resource competition among different jobs on each processor, deadline requirements may be violated. The paper introduces a distributed approach to assigning local deadlines to the jobs on each processor. The approach leads to improved schedulability results by considering disparate workloads among the processors due to competing jobs having different paths. Simulation results based on randomly generated workloads indicate that the proposed approach outperforms existing work in terms of both the number of feasible task sets (between 22% and 75%) and the number of feasible jobs (between 57% and 46%).**

I. Introduction

Distributed real-time systems are widely employed in cyber-physical applications such as vehicle control and multimedia communication (e.g., [7], [14], [23]). Such systems typically require that jobs be executed on a chain of processors and be completed within some end-to-end deadlines. Resource competition among the jobs on a shared processor could severely increase job response times, potentially resulting in end-to-end deadline misses. Therefore, it is important to properly manage job execution patterns on each processor in order to guarantee the timing requirements of tasks in a distributed real-time system. Assigning job priorities locally on each processor is an efficient way to achieve such management.

A number of recent papers have investigated the job-priority assignment problem for distributed real-time systems. Most of these solutions assign a local deadline for each job executing on each processor. The local deadlines then dictate the job priorities according to the earliest-deadline-first (EDF) scheduling policy. Some of the local-deadline assignment approaches follow the general idea of dividing the end-to-end deadline of a job into segments to be used as local deadlines by the processors that will execute the job. The division may depend on the number of processors that the job traverses [12], [22] or execution time distribution of the job among the processors [3]. In another local-deadline assignment approach [19], the local deadline of each task is assumed to be given and the local deadline of each job is derived on-line based on the job completion time in the preceding processors. Since none of the above mentioned methods consider work-loads contributed by different jobs on a processor, they cannot guarantee that jobs on a shared processor are schedulable, which in turn may lead to eventual end-to-end deadline misses. To ensure the feasibility of the jobs on each processor, the work in [13] employs the feasibility condition from [1] to assign local deadlines to tasks on each processor in an off-line, iterative manner. However, this approach is not only time consuming but also assumes that the tasks on each processor are synchronized, which can be quite pessimistic. The authors in [20] propose a local-deadline assignment scheme aiming to minimize processor resource requirements but this approach focuses only on a single task.

Instead of assigning local deadlines, Jayachandran and Abdelzaher propose to assign priorities to jobs based on the job's absolute end-to-end deadlines ([9], [11]). This approach works well when the workload of each processor is relatively low. For relatively high workloads, such an approach can be ineffective, particularly when different jobs take different paths of execution and workloads on processors vary significantly.

A topic closely related to the job-priority assignment problem is schedulability analysis of tasks in distributed real-time systems as such analysis is critical for evaluating the quality of an assignment. Some work (e.g., [16], [18]) on this topic presented algorithms to compute the time demand bound function of tasks scheduled by the EDF policy. Some other work (e.g., [8], [10], [15]) proposed methods to compute the worst-case response times for systems scheduled by the fixed priority or EDF policy. The approach in [9], [11] transforms the distributed real-time system schedulability test into a uniprocessor schedulability test. It is well known that feasibility-analysis based methods are generally time consuming and not suitable for on-line use. Furthermore, the method in [8] is for fixed priority scheduling, which may under-utilize resources compared to EDF. The schedulability test proposed in [9], [11] does not work well if transactions have different execution paths and the ratios of the end-to-end deadlines to periods for some transactions are much larger than the number of processors along the execution paths.

In this paper, we present a distributed approach which combines local-deadline assignment with feasibility analysis such that the resulting deadline assignment is guaranteed to be schedulable. Our approach formulates the local-deadline assignment problem as a mathematical programming problem of maximizing the minimum time slack among all the jobs executed on each processor. In order to account for job interferences on each processor, we introduce a novel transformation of the necessary and sufficient condition proposed in [5], [6]. Our mathematical programming formulation addresses the shortcomings of existing work by considering resource competition among all the jobs on a shared processor and coordinating the local deadline distribution of a job at different stages along the job's execution path.

We further introduce an on-line, iterative technique to efficiently and effectively solve the mathematical programming based local-deadline assignment problem. Our proposed algorithm, with a worst-case time complexity of $O(N^3 \cdot M)$ (where N and M is the total number of jobs and the total number of processors, respectively), is guaranteed to find a feasible solution if such a solution exists. In practice, the solution search process requires only a few steps. Given its distributed nature, the algorithm readily adapt to dynamic changes in job execution times and execution paths. Furthermore, since it only needs local execution information, the algorithm avoids the overhead of global time synchronization. The observations made in the optimality proofs reveal some interesting properties (such as prediction of idle-time existence in a job-set execution) for the type of special job sets used in our algorithm and can be applicable to similar feasibility studies.

We have conducted simulation-based studies of our algorithm and compared it with two existing representative methods, JA [9], [11] and BBW [3]. Our studies show that, for systems where processor workloads are somewhat balanced, our approach can find on average 75% and 22% more feasible solutions than JA and BBW, respectively. For systems where processor workloads vary noticeably, our approach leads to, on average, 57% to 46% more solutions than JA and BBW, respectively. Furthermore, for balanced workloads, JA and BBW drop 325% and 225% more jobs on average than our approach, respectively. For imbalanced workloads, the averages are 89% and 379%, respectively. These results show that our approach indeed leads to improved quality of service for distributed real-time systems.

The rest of the paper is organized as follow. Section II provides the system model and motivations for our work. Section III presents our approach to solve the local-deadline assignment problem. Experimental results are presented and discussed in Section IV, and Section V concludes the paper.

II. Preliminaries

Below, we first introduce the needed notation and scheduling properties. We then give motivations for our work.

A. System Model

We consider a distributed system, where a set of real-time tasks arrive either periodically or aperiodically and require execution on a sequence of processors. Since our focus is on on-line distributed deadline assignment methods, we only consider individual task instances, i.e., jobs, without specific assumptions about task periodicity. A job J_i is released at time R_i, needs to be executed on M_i processors (also referred to as M_i stages), and must be completed by its absolute end-to-end deadline, D_i, The segment of J_i running at stage k ($k = 1, \cdots, M_i$) is denoted as $J_{i,k}$, whose worst-case execution time is $C_{i,k}$. Each $J_{i,k}$ is associated with an absolute release time $r_{i,k}$ and absolute deadline $d_{i,k}$, both of which are to be determined during the local-deadline assignment process. Note that $r_{i,1} = R_i$ and we assume that $r_{i,k} = d_{i,k-1}$.

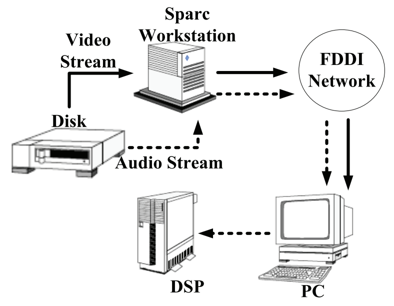

Fig. 1. A teleconferencing application consisting of video and audio streams [4].

Similar to the execution models in [9], [11], we assume that there is a total execution order among the processors in the given distributed system. That is, if $x < y$, processor V_x should appear before processor V_y in any job's execution path containing processors V_x and V_y. We refer to any processor V_x having the execution order earlier (later) than V_y as an *upstream* (*downstream*) processor of V_y. Processor V_x has a set $\Omega(V_x)$ of jobs that traverse it (i.e., are executed on it). We use $J_{i,k(i,x)} \in \Omega(V_x)$ to indicate that the k-th stage of job J_i is executed on processor V_x. (Function $k(i,x)$ returns the integer denoting the stage index when J_i executes on V_x.) The above execution models can be found in many signal/data processing applications. For example, Figure 1 shows a teleconferencing application presented in [4], [14]. This application consists of video and audio stream processing, J_1 and J_2, where the video stream and the audio stream have almost the same path except that the audio stream completes its processing in the DSP component after finishing its execution on a PC. The processor execution order for this application can thus be represented as V_1 (disk), V_2 (Sparc Workstation), V_3 (FDDI Network), V_4 (PC) and V_5 (DSP), $J_{1,k(1,1)} = J_{1,1}$, $J_{2,k(2,1)} = J_{2,1}$, etc.

One way to meet the jobs' end-to-end deadlines is to assign local job deadlines such that all the jobs on every processor are schedulable and that the local deadlines at the jobs' last stage are less than or equal to the respective end-to-end deadlines. Since the execution model enforces an execution order, it is important for upstream processors to not overuse their shares of slacks and to leave enough time for later stages. Specifically, we define the time slack of $J_{i,k}$ as the difference between the end-to-end deadline relative to $r_{i,k+1}$ and the sum of the execution times from stage $k+1$ to stage M_i, i.e.,

$$s_{i,k} = D_i - r_{i,k+1} - \sum_{m=k+1}^{M_i} C_{i,m}. \tag{1}$$

The time slack gives information on the longest delay that a job can endure from the current stage to the job's final stage without violating its end-to-end deadline. Maximizing the time

TABLE I

A Motivating Example Containing Two Jobs that Traverse Four Processors

Processor Name	Job τ_1		Job τ_2		Local Deadline Assignment BBW / OLDA		Response Time BBW / JA / OLDA	
	Execution Time	End-to-End Deadline	Execution Time	End-to-End Deadline	Job τ_1	Job τ_2	Job τ_1	Job τ_2
Processor V_1	100	N/A	70	N/A	111 / 100	90 / 170	170 / 170 / 100	70 / 70 / 170
Processor V_2	200	N/A	430	N/A	331 / 300	663 / 730	370 / 700 / 300	700 / 500 / 730
Processor V_3	100	N/A	100	N/A	441 / 400	797 / 830	470 / 800 / 400	800 / 600 / 830
Processor V_4	600	1100	100	930	1100 / 1100	930 / 930	1170 / 1400 / 1100	900 / 700 / 930

slack of each job on any processor gives the best opportunity to satisfy the end-to-end deadline requirements.

We assume that EDF is used on each processor since it is optimal in terms of meeting job deadlines for a single processor. Each job should complete execution at the last stage within its end-to-end deadline. However, preemptions on processors with different workloads along the job's execution path may cause some jobs to have long response times at some stages and eventually miss their end-to-end deadlines. Our problem, then, is to minimize the response times of jobs at each stage by intelligently assigning local deadlines to them. A necessary and sufficient condition for schedulability under EDF on a uniprocessor is restated below with proper notation.

Theorem 1: **[5], [6]** Job set $\Omega(V_x)$ can be scheduled by EDF if and only if $\forall J_{i,k(i,x)}, J_{j,k(j,x)} \in \Omega(V_x)$, $r_{i,k(i,x)} \leq d_{j,k(j,x)}$,

$$d_{j,k(j,x)} - r_{i,k(i,x)} \geq \sum_{\substack{J_{r,k(r,x)},\\ r_{r,k(r,x)} \geq r_{i,k(i,x)},\\ d_{r,k(r,x)} \leq d_{j,k(j,x)}}} C_{r,k(r,x)}. \qquad (2)$$

B. Motivations

We use a simple distributed real-time system to illustrate the deficiencies of existing approaches in terms of satisfying the real-time requirements. The example application contains 2 jobs; their computation times and end-to-end deadlines are shown from columns 2 to 5 in Table I. Both jobs, J_1 and J_2, are released at time 0 onto processor V_1, and traverse processors V_1, V_2, V_3 and V_4 sequentially.

We consider two representative priority assignment methods: JA [9], [11] and BBW [3]. JA, introduced by Jayachandran and Abdelzaher, belongs to the job-level fixed priority based approaches. It assigns a priority to a job according to its absolute end-to-end deadline and this priority is fixed across all processors executing this job [9], [11]. Although this priority assignment is extremely efficient and incorporates the real-time requirement of each individual job, it does not take into account the fact that different jobs may have different execution paths and some processors may have higher utilization than others. Hence, it may cause a job with a small time slack to be preempted by another job with a large time slack on some processors, and eventually result in deadline misses. BBW, proposed by Buttazzo, Bini and Wu, is an end-to-end deadline partitioning based method. It assigns local deadlines to the job on each processor by partitioning its end-to-end deadline in proportion to the job's executions times at different stages [3]. The disadvantage of BBW is that some jobs may be preempted on an upstream processor for a long time due to its long local deadline, and then fail to catch up in time to meet their end-to-end deadlines later.

In the motivating example, the local deadlines assigned by BBW (and the resultant job response times) at each processor are indicated by the first value in columns 6 and 7 (and the first value in columns 8 and 9) of Table I. The resultant job response times at each processor obtained by JA are shown in the second value in columns 8 and 9. For example, under BBW, the local absolute deadline of job J_1 on processor V_1 equals 111 time units and the response time is 170 time units. With JA, job J_1 completes its execution at stage 4 (the last stage) at time 1400, which is much longer than its end-to-end deadline. BBW performs a little better than JA in reducing the response time of job J_1, but still causes J_1 to miss its end-to-end deadline. Since JA ignores the workload of each job on different processors along the job's execution path, it may assign a low priority to a job with large computation times in the remaining stages and cause the job to miss its end-to-end deadline. On the other hand, BBW ignores the resource competition among different jobs on a shared processor and causes both the local and end-to-end deadlines to be missed. If an alternative local deadline assignment method can consider both the workloads along a job's execution path and resource competition among different jobs on a shared processor, adopting such a method may result in meeting the deadline requirements of both jobs J_1 and J_2. We will present one such method, OLDA (On-line Distributed Algorithm), in next section. The new local deadlines obtained by OLDA are represented by the second values in columns 6 and 7 and the resultant response times are as given by the third values in columns 8 and 9 in Table I. It is clear that this local deadline assignment is able to meet the end-to-end deadlines of both jobs.

III. Our Approach

As shown in the last section, the probability that tasks meet their end-to-end deadlines can be greatly increased if appropriate local deadlines are assigned to the jobs on different processors. Although it is possible to accomplish local-job deadline assignment in a global manner using mathematical programming or dynamic programming, such approaches incur high computation overhead and are not suitable for on-line use.

We adopt a distributed, on-line approach to determining local job deadlines on each processor. On a high level, our approach works as follows. Every time a new job arrives at processor V_x, new deadlines are assigned for both the newly arrived job and current jobs (which may be partially executed) on V_x. If a feasible deadline assignment is not found, the job with the maximum accumulative unfinished execution time in the remaining stages (among all the jobs in V_x) will be eliminated from further processing.

The key to make the above generic distributed approach effective is the design of an appropriate local deadline assignment algorithm to be run on each processor. Below, we focus our discussions on how to achieve such a design. We first present a mathematical programming based formulation for local deadline assignment. Although the resultant problem can be solved directly by a solver, the time overhead can be rather high for on-line use. Based on the mathematical programming formulation, we introduce a efficient and optimal algorithm for solving the local deadline assignment problem.

A. Mathematical Programming Formulation

In order to ensure that a local deadline assignment lead to a feasible schedule on a processor, a deadline assignment approach should consider resource competition among different jobs on that processor. Furthermore, such as approach should leave as much time slack (see the definition in (1)) as possible for later stages to help satisfy the end-to-end deadline requirements. More specifically, our goal is to determine the local deadline $d_{i,k}$ for job $J_{i,k}$ such that the end-to-end deadline of J_i is met, the job set $\Omega(V_x)$ on processor V_x is schedulable, and (1) is maximized. We capture the problem as a constrained optimization problem given below:

$$\text{max:} \quad \min_{J_{i,k(i,x)}\in\Omega(V_x)} \left\{ D_i - d_{i,k(i,x)} - \sum_{m=k(i,x)+1}^{M_i} C_{i,m} \right\} \tag{3}$$

$$\text{s.t.} \quad r_{i,k(i,x)} + C_{i,k(i,x)} \le d_{i,k(i,x)} \le D_i - \sum_{m=k(i,x)+1}^{M_i} C_{i,m}, \quad \forall J_{i,k(i,x)} \in \Omega(V_x) \tag{4}$$

$$d_{i,k(i,x)} - r_{j,k(j,x)} \ge \sum_{\substack{J_{h,k(h,x)}\in\Omega(V_x),\\ r_{h,k(h,x)}\ge r_{j,k(j,x)},\\ d_{h,k(h,x)}\le d_{i,k(i,x)}}} C_{h,k(h,x)}, \quad \forall J_{j,k(j,x)}, J_{i,k(i,x)} \in \Omega(V_x). \tag{5}$$

Since it is ambiguous to require maximizing the time slacks of all jobs on a processor, we resort to the objective of maximizing the minimum time slack among all the jobs executed on V_x (see (3)). Constraints (4)–(5) are used to guarantee schedulability on each processor. Specifically, constraint (4) bounds the local deadline of jobs execution on V_x by the earliest completion time of the job (left side of (4)) and the latest start time of the immediate next stage (right side of (4)). Constraint (5) is simply a restatement of (2).

An astute reader would notice that the above constraint optimization problem formulation cannot be straightforwardly solved by a mathematical programming solver. This is because the actual terms to be included in the summation on the right hand side of constraint (5) depend on local deadlines which are themselves decision variables. To overcome this challenge, we introduce an observation in the following lemma, which can be used to convert constraint (5) to a form readily solvable by a mathematical programming solver (The proof for the lemma can be found in the appendix). The essence of the observation is that the EDF schedulability of a job set can be checked by examining certain behavior of all the job subsets in the job set. This observation also plays a key role in developing the efficient algorithm to be presented later.

Lemma 1: Given job set $\Omega(V_x)$ to be executed on processor V_x according to the EDF policy, let $\omega(V_x)$ represent any subset of $\Omega(V_x)$. $\Omega(V_x)$ is schedulable if and only if

$$\max_{J_{i,k(i,x)}\in\omega(V_x)}\{d_{i,k(i,x)}\} - \min_{J_{i,k(i,x)}\in\omega(V_x)}\{r_{i,k(i,x)}\} \ge \sum_{J_{i,k(i,x)}\in\omega(V_x)} C_{i,k(i,x)}, \quad \forall\omega(V_x)\subseteq\Omega(V_x). \tag{6}$$

Based on Lemma 1, we can substitute constraint (5) in our optimization problem with constraint (6). In this new constraint, the number of terms in the max function only depends on the corresponding job subset. Therefore, the resulting problem specified by (3), together with (4) and (6), can be solved by a mathematical programming solver. We will discuss the use of one such solver in Section IV.

B. On-Line Distributed Algorithm (OLDA)

In this section, we present OLDA, our local deadline assignment algorithm, which exactly and efficiently solves the optimization problem given in (3), (4) and (6). There are multiple challenges in designing OLDA. The most obvious difficulty is how to avoid checking the combinatorial number of subsets of $\Omega(V_x)$ in constraint (6). Another challenge is how to maximize the objective function in (3) while ensuring job schedulability and meeting all end-to-end deadlines. Below, we discuss how our proposed algorithm overcomes these challenges, and describe the algorithm in detail along with some theoretical foundations behind it. Unless explicitly noted, the deadline of a job in this section always means the local deadline of the job on the processor under consideration.

One key idea in OLDA is to construct a unique subset for a given job set $\Omega(V_x)$. Using this job subset, OLDA can determine the local deadline of at least one job among all the jobs in $\Omega(V_x)$ such that this local deadline is guaranteed to belong to an optimal solution for the problem given in (3), (4) and (6). We refer to this unique job subset of $\Omega(V_x)$ as the *base subset* of $\Omega(V_x)$ and it has the following property.

Property 1: If $\omega^*(V_x)$ is a base subset of $\Omega(V_x)$, then

$$\begin{aligned} d_{*,k(*,x)} &= \min_{J_{i,k(i,x)}\in\omega^*(V_x)}\{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)}\in\omega^*(V_x)} C_{i,k(i,x)} \ge \min_{J_{i,k(i,x)}\in\omega(V_x)}\{r_{i,k(i,x)}\} \\ &+ \sum_{J_{i,k(i,x)}\in\omega(V_x)} C_{i,k(i,x)}, \qquad \forall\omega(V_x)\subseteq\Omega(V_x). \end{aligned}$$

For a given base subset, determining which job to assign a deadline and what value the deadline should have constitutes another key idea in OLDA. Recall that our optimization goal

Algorithm 1 OLDA($\Omega(V_x)$)

```
1: $\Omega(V_x) = Sort\_Non\_Dec\_Release\_Time(\Omega(V_x))$
2: $Upper\_Bound = Comp\_Local\_Deadline\_UB(\Omega(V_x))$ //Compute the upper bound on local deadline of each job in $\Omega(V_x)$
3: **while** ($\Omega(V_x) \neq \emptyset$) **do**
4:    $\omega(V_x) = \Omega(V_x)$
5:    $\omega^*(V_x) = \Omega(V_x)$
6:    $Max\_Deadline = 0$
7:    **while** ($\omega(V_x) \neq \emptyset$) **do**
8:       $Temp\_Deadline = \min_{J_{i,k(i,x)} \in \omega(V_x)} \{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)} \in \omega(V_x)} C_{i,k(i,x)}$
9:       **if** ($Max\_Deadline \leq Temp\_Deadline$) **then**
10:          $Max\_Deadline = Temp\_Deadline$
11:          $\omega^*(V_x) = \omega(V_x)$
12:       **end if**
13:       $\omega(V_x) = Remove\_Earliest\_Released\_Job(\omega(V_x))$ //Remove the job with the earliest release time
14:    **end while**
15:    $Max\_upper\_bound = 0$
16:    **for** (each $J_{i,k(i,x)} \in \omega^*(V_x)$) **do**
17:       **if** ($Upper\_bound[J_{i,k(i,x)}] > Max\_upper\_bound$) **then**
18:          $Max\_upper\_bound = Upper\_bound[J_{i,k(i,x)}]$
19:          $J_{*,k(*,x)} = J_{i,k(i,x)}$
20:       **end if**
21:    **end for**
22:    **if** ($Max\_upper\_bound < Max\_Deadline$) **then**
23:       $Drop\_Selected\_Job(\Omega(V_x))$ //Drop the job from $\Omega(V_x)$, which has the maximum unfinished execution times at the remaining stages
24:       **return** $\emptyset$
25:    **else**
26:       $d_{*,k(*,x)} = Max\_Deadline$
27:       $\Omega(V_x) = \Omega(V_x) - J_{*,k(*,x)}$
28:    **end if**
29: **end while**
30: **return** $\{d_{i,k(i,x)}\}$
```

is to maximize the job time slacks. Hence, we select this job based on the end-to-end deadlines and remaining execution times of all the jobs in the base set. We refer to the selected job as the base job and it has the following property.

Property 2: If $J_{*,k(*,x)} \in \omega^*(V_x)$ is a base job for job set $\Omega(V_x)$, then

$$D_* - \sum_{m=k(*,x)+1}^{M_*} C_{*,m} \geq D_i - \sum_{m=k(i,x)+1}^{M_i} C_{i,m} \quad \forall J_{i,k(i,x)} \in \omega^*(V_x),$$

Based on the above two properties, for the remaining jobs in the given job set, OLDA iteratively constructs base subsets and assigns local deadlines to the corresponding base jobs.

Algorithm 1 summarizes the main steps in OLDA. (Recall that this algorithm is used by each processor in a distributed manner, and the pseudocode is given for processor V_x.) OLDA starts by sorting the given jobs in a non-decreasing order of their release times (Line 1) and computing the upper bound on the local deadline for each job (Line 2). The upper bound on the local deadline of a job is the value beyond which the job will definitely miss its end-to-end deadline. Note that this is simply the values used in Property 2 for the base job. The algorithm enters the main loop spanning from Line 3 to Line 29. The first part in the main loop (Line 4 to Line 14) constructs the base subset for the given job set according to Property 1 and computes the desired deadline value ($Max\_Deadline$). The second part of the main loop (Line 15 to Line 21) makes use of Property 2 to find the base job in the base subset. If the desired deadline value is smaller than or equal to the local deadline upper bound of the base job (as shown in Line 2), the third part of the main loop assigns this value to the base job (denoted by $d_{*,k(*,x)}$) as its local deadline (Line 26), removes $J_{*,k(*,x)}$ from $\Omega(V_x)$ (Line 27), and repeats the process. In the case where the desired deadline value is larger than the upper bound on the local deadline of the base job, the job with the largest unfinished execution times at the remaining stages is removed from the original set $\Omega(V_x)$ and OLDA will restart from this new job set.

It is worth noting that OLDA only requires information that is local to processor V_x (such as job release times) and information that is known upon a job's release (such as the path of jobs and the upper bound on the local absolute deadline of each job. The information that becomes available upon jobs' releases can be relayed from an upstream processor to a downstream processor. (This is possible since there is a total execution order among the processors.) Therefore, OLDA does not require global time synchronization.

We claim that OLDA solves the optimization problem given by (3), (4) and (6). That is, if there exists a solution to the problem, OLDA always find it. Furthermore, if there is no feasible solution to the problem, OLDA always identifies this case. To support our claim, we first show that the local deadline assignment made by OLDA (when no jobs are dropped) satisfies the constraints in (4) and (6). This is given in the following two lemmas. For readability, we have put all the proofs in this section (except for the proof for Lemma 3, which is straightforward) in the appendix.

Lemma 2: Given job set $\Omega(V_x)$, let $d^*_{i,k(i,x)}$ be the local deadline assigned by OLDA to $J_{i,k(i,x)} \in \Omega(V_x)$. Then

$$r_{i,k(i,x)} + C_{i,k(i,x)} \leq d^*_{i,k(i,x)} \leq D_i - \sum_{m=k(i,x)+1}^{M_i} C_{i,m}, \quad \forall J_{i,k(i,x)} \in \Omega(V_x) \quad (7)$$

Lemma 3: Given job set $\Omega(V_x)$, let $d^*_{i,k(i,x)}$ be the local deadline assigned by OLDA to $J_{i,k(i,x)} \in \Omega(V_x)$. We have

$$\max_{J_{i,k(i,x)} \in \omega(V_x)} \{d^*_{i,k(i,x)}\} - \min_{J_{i,k(i,x)} \in \omega(V_x)} \{r_{i,k(i,x)}\} \geq \sum_{J_{i,k(i,x)} \in \omega(V_x)} C_{i,k(i,x)}, \quad \forall \omega(V_x) \subseteq \Omega(V_x). \quad (8)$$

Proving that the local deadline assignment made by OLDA indeed optimizes the objective function in (3) requires analyzing the relationship among the jobs' time slacks. Since OLDA assigns job local deadlines by identifying the base job in each base subset, a special property that the base subset possesses greatly simplifies the analysis process. Lemma 4 below summarizes this property.

Lemma 4: Let $\omega^*(V_x)$ be a base subset of job set $\Omega(V_x)$ and $r^* = \min_{J_{i,k(i,x)} \in \omega^*(V_x)}\{r_{i,k(i,x)}\}$. Under the work-conserving EDF policy, processor V_x never idles once it starts to execute the jobs in $\omega^*(V_x)$ at r^* until it completes all the jobs in $\omega^*(V_x)$. In addition, the busy interval corresponding to executing the jobs in $\omega^(V_x)$ is $[r^*, r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}]$.

Based on Lemma 4, the optimality of the deadline assignment made by OLDA can be proved. This conclusion is stated in the following lemma.

Lemma 5: Given job set $\Omega(V_x)$, let $d^*_{i,k(i,x)}$ be the local deadline assigned to each $J_{i,k(i,x)} \in \Omega(V_x)$ by OLDA. Then $d^*_{i,k(i,x)}$ maximizes the function given in (3).

To show that OLDA always identifies the case where there is no feasible solution to the optimization problem, we observe that OLDA always finds a local deadline assignment without dropping any jobs if there exists a feasible solution to the optimization problem. This is stated in the following lemma.

Lemma 6: Given job set $\Omega(V_x)$, if there exists $d_{i,k(i,x)}$ for every $J_{i,k(i,x)} \in \Omega(V_x)$ that satisfies (4) and (6), OLDA always finds a feasible local deadline assignment for every $J_{i,k(i,x)} \in \Omega(V_x)$.

Based on Lemmas 2, 3, 5, and 6, we have the following theorem.

Theorem 2: In $O(|\Omega(V_x)|^3)$ time, OLDA returns a set of local deadlines if and only if there exists a solution to the optimization problem specified in (4), (6), and (3). Furthermore, the returned set of local deadlines is a solution to the optimization problem.

We omit the actual proof for Theorem 2 and only discuss its time complexity. The time complexity of OLDA is dominated by the main `while` loop starting at Line 3. (Refer to Algorithm 1.) Inside the `while` loop, the most time consuming operations appears in the inner `while` loop from Line 7 to Line 14. Every time OLDA computes the local absolute deadline of a job, it considers $|\Omega(V_x)|$ number of subsets. Furthermore, the number of jobs in $\Omega(V_x)$ is always reduced by 1 in each iteration. Hence, OLDA considers $\frac{|\Omega(V_x)| \cdot |\Omega(V_x)+1|}{2}$ number of subsets of $\Omega(V_x)$ in total, instead of a combinatorial number of them. In addition, it takes $|\omega^*(V_x)|$ iterations to compute $\min_{J_{i,k(i,x)} \in \omega^*(V_x)}\{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}$ for each subset $\omega^*(V_x)$. Therefore, the time complexity of OLDA when considering a set of jobs running on processor V_x is $O(|\Omega(V_x)|^3)$.

IV. Experimental Results

In this section, we discuss the the performance and efficiency of our proposed algorithm using randomly generated job sets. We also compare our algorithm with a job-level fixed priority based method, JA in [9], [11], and an end-to-end deadline partitioning method, BBW in [3]. We further illustrate the use of our algorithm in a real-world application. In the rest of the section, we first describe our simulation setup, then summarize simulation results based on the randomly generated job sets, and finally present a case study.

A. Simulation Setup

To evaluate our proposed algorithm (Section III-B), we used two different benchmarks consisting of randomly generated task sets in order to evaluate two different processor loading scenarios. Each benchmark contains 100 randomly generated task sets of 50 tasks each for 10 different system utilization levels $(400\%, 425\%, \ldots, 625\%)$, for a total of 1,000 task sets. There are 8 processors in total and each task randomly passes through 4 to 6 stages. Task periods ranged from 1,000 to 10,000 time units. We used the UUnifast algorithm [2] to generate the total execution time of each task. After the call to the UUnifast algorithm [2], and based on the actual number of stages M_i for each task τ_i, the set of processors used by task τ_i is randomly assigned, along with the execution time $C_{i,k}$ at each stage k. Each task set is generated with the guarantee that the total utilization at each processor is no larger than 1.

For the first benchmark, the execution time of a job at each stage is randomly distributed. As a result, processor loads tend to be balanced. As a stress test, the second benchmark represents a somewhat imbalanced workload distribution among the processors. It is generated in such a way that the first few stages as well as the last few stages are more heavily loaded. This benchmark is designed to test the usefulness of considering severe resource competition among different tasks on a given processor in meeting end-to-end deadlines.

To ensure a fair comparison for the three algorithms under consideration, we made some modifications to JA and BBW. The original version of both JA and BBW require global time synchronization. We removed this requirement by implementing JA and BBW on-line on each processor. All algorithms were implemented in C++ and experimental data were collected on two quad-core 2.3 GHz AMD Opteron processors with Red Hat Linux 4.1.2-50.

B. Results for Randomly Generated Benchmarks

We discuss the following comparisons in this section. First, we compare the number of task sets in which all jobs meet their end-to-end deadlines using OLDA, BBW and JA. Second, for the task sets where at least one end-to-end deadline is missed or a feasible local-deadline assignment is not found, we compare the job drop rates (the ratio between the number of jobs dropped and the number of jobs released in the system) of the algorithms. Third, we assess the time overhead of OLDA.

In the first experiment, we compare the percentage of feasible task sets (over the 100 task sets at each utilization level) found by our algorithm, as opposed to those found by JA and BBW for both the balanced and imbalanced workload scenarios. The results are summarized in Figures 2 and 3,

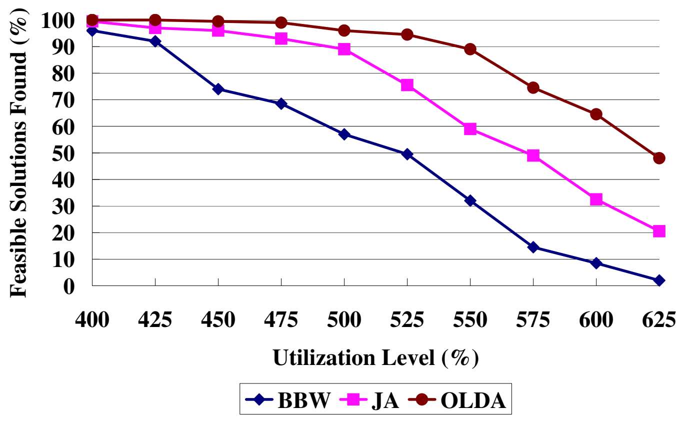

Fig. 2. Comparison of OLDA, JA and BBW in terms of percentage of feasible task sets found for balanced workloads.

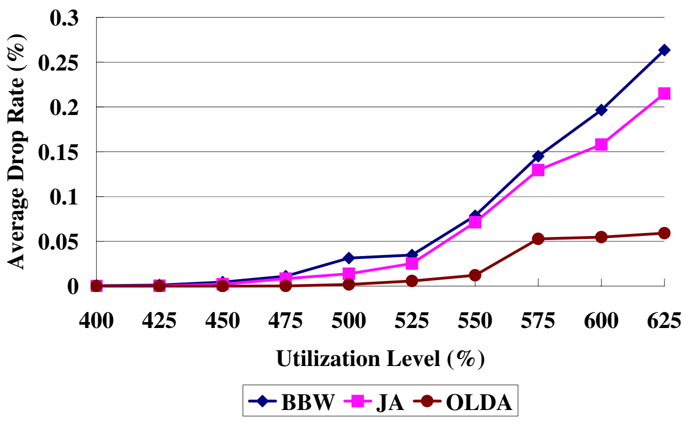

Fig. 4. Comparison of OLDA, JA and BBW in terms of average drop rate for balanced workloads.

respectively. It is clear from the plots that OLDA can find far more feasible sets than the other methods. Specifically, for balanced workloads, OLDA leads to 75% and 22% on average (and up to 2,300% and 134%) more feasible task sets than BBW and JA, respectively. For imbalanced workloads, OLDA results in 57% and 46% on average (and up to 184% and 650%) more feasible task sets than BBW and JA, respectively. Observe that OLDA performs much better than existing techniques at high utilization levels where there are more jobs in the system. We want to point out that sometimes OLDA may not be able to find a feasible solution even though such solutions indeed exists, since OLDA finds local job deadlines for each processor independently instead of using a time-consuming global approach. However, for balanced workloads, OLDA can find on average 98% and 99% of the feasible task sets found by BBW and JA, respectively. For imbalanced workloads, OLDA can find on average 88% and 99% of the feasible task sets found by BBW and JA, respectively. These results demonstrate that OLDA not only finds more feasible task sets than BBW and JA, but also solves most of the problems that BBW and JA solve.

In the second experiment, we compare the average job drop rates of infeasible task sets when using different algorithms for both balanced and imbalanced workloads. The drop rate of an infeasible task set is computed as the ratio between the number of drop jobs and the total number of jobs released to the system. (A job is dropped either because no local deadline assignment can be found for the job set on a processor by OLDA or the job's end-to-end deadline is missed using BBW and JA.) The job drop rates for the three algorithms for the balanced and imbalanced workloads are shown in Figures 4 and 5, respectively. It is clear from the plots that OLDA drops much fewer jobs than the two other methods. For balanced workloads, BBW and JA drop 325% and 225% more jobs on average than OLDA, respectively. For imbalanced workloads, the averages are 89% and 379%, respectively.

To examine whether OLDA is suitable for on-line local deadline assignments, we compare the number of cycles required by OLDA against those of JA and BBW. For the benchmark with balanced workloads, OLDA requires on average 3.64 and 23.30 times more cycles per task set (with 50 tasks) than BBW and JA, respectively. For the benchmark with imbalanced workloads, which is more difficult to solve, OLDA needs about 4.83 and 30.79 times more cycles per task set than BBW and JA, respectively. Although OLDA has a longer running time than both BBW and JA, the average number of

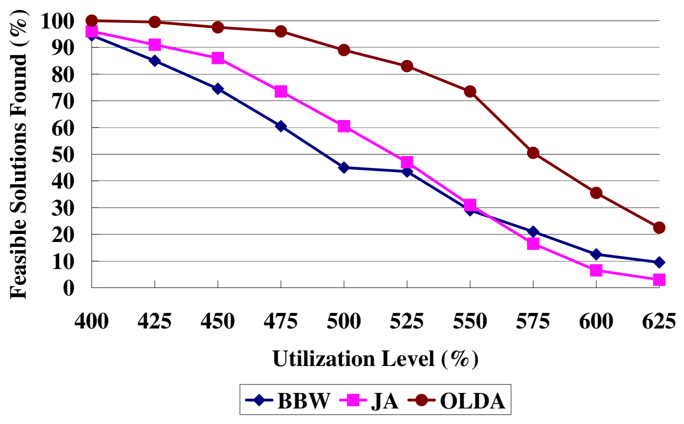

Fig. 3. Comparison of OLDA, JA and BBW in terms of percentage of feasible task sets found for imbalanced workloads.

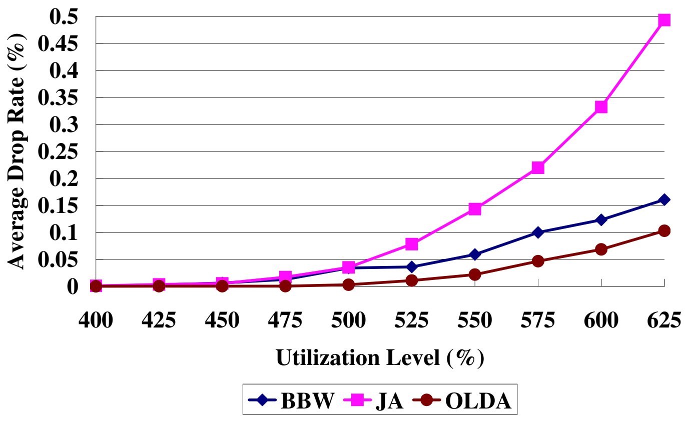

Fig. 5. Comparison of OLDA, JA and BBW in terms of average drop rate for imbalanced workloads.

TABLE II

A CASE STUDY OF A FLIGHT CONTROL SYSTEM

Trans.	Execution Time									End2End	Dropped Job Num.		End2End	Dropped Job Num.
Name	AH	NV	FC	BS	FG	AP	SV	PF	Period	Deadline	OLDA / JA / BBW	Period	Deadline	OLDA / JA / BBW
FCP	0	0	15	29	10	15	0	10	500	450	0 / 0 / 0	120	120	0 / 0 / 0
PAA	10	0	0	16	15	20	10	0	100	100	0 / 0 / 0	72	72	0 / 180 / 0
NIP	0	10	0	14	20	0	0	0	250	200	0 / 0 / 0	75	75	0 / 0 / 30

cycles required to process a job is about 315 and 371 cycles for the balanced workload and imbalanced workload, respectively. (Note that the time overhead for JA and BBW is negligible.) Such runtime overhead is tolerable in systems where real-time jobs are computationally demanding, e.g. in avionics and automotive control applications [9], [17] where jobs usually require hundreds of thousands or millions of cycles to execute.

To see how well OLDA fares compared with a mathematical programming solver, we randomly selected a few benchmarks and compared the solutions found by OLDA to the ones found by Loqo [21], a system for solving smooth constrained optimization problems. The comparisons support our earlier claim that OLDA always find an optimal solution to the problem stated in (4), (6), and (3) whenever a feasible solution exists. Furthermore, the execution time of OLDA is about 10,000$\times$ shorter than Loqo.

C. Case Study of a Flight Control System

Using a large number of randomly generated task sets, we have shown that OLDA outperforms existing methods. However, it is important to quantify the performance of OLDA under a real-world workload. We use a simplified flight control system, similar to the one in [9], to illustrate how OLDA adapts to changing requirements on-line to guarantee the end-to-end deadlines of jobs in an actual distributed real-time system. The system contains 3 periodic tasks and 8 heterogeneous processors. Flight control processing (FCP) task reads input commands from Flight Control Processor (FC), processes them on Flight Guidance System (FG) and Auto-Pilot (AP) sequentially, and displays the results on Primary Flight Display (PF). Pitch adjustment actuation (PAA) task receives periodic sensor readings from Attitude and Heading Reference System (AH), processes the information on FG and AP sequentially, and sends control signals to Elevator Servo (SV). Navigation information processing (NIP) task periodically receives sensor readings from Navigation Radio (NV) and processes them on Flight Guidance System (FG). All input commands and sensor readings reach FG through a Bus (BS). The task execution times on the processor are given in columns 2 to 10 and the task end-to-end deadlines in column 11 of Table II, where the time values are in milliseconds.

We simulated the application for the time interval $[0, 54000ms]$ when applying OLDA, JA and BBW, and found that no job misses its end-to-end deadline. Now, assume that at some time interval, the airplane encounters some emergency, such as air turbulence or a mechanical malfunction. For security, the periods and end-to-end deadlines of the three tasks are decreased, as shown in columns 13 and 14 of Table II. If the previous local deadline assignments generated by BBW are reused for the tasks, 60 jobs out of the 1920 released jobs will miss their end-to-end deadlines. (Note that neither JA nor OLDA can reuse local job deadlines when task periods change.) Suppose we apply JA and BBW in response to the workload and deadline changes, 180 and 30 jobs are dropped from the PAA and NIP tasks, respectively (see the second and third numbers in column 15 of Table II). In contrast, applying OLDA leads to all jobs meeting their end-to-end deadlines.

V. SUMMARY AND FUTURE WORK

We presented a novel distributed local deadline assignment approach to guarantee task end-to-end deadlines in a distributed real-time system. Our algorithm has the following features: (i) it is guaranteed to find an optimal solution if one exists, (ii) it is effective even when different tasks have different paths and workloads on different processors are dissimilar, (iii) it avoids the overhead of global time synchronization and is efficient enough for online use, and (iv) it can adapt to dynamic changes in the system. Our future plans include generalizing the proposed algorithm to handle situations where processors have no execution order and implementing the algorithm in a real-time operating system.

VI. ACKNOWLEDGEMENT

This work is supported in part by NSF under grants CNS-0931195 and CCF-0702761.

APPENDIX

Proof for Lemma 1: We first prove the "if" part. Assume that $\forall \omega(V_x) \subseteq \Omega(V_x)$, constraint (6) holds, i.e.,

$$\max_{J_{h,k(h,x)} \in \omega(V_x)} \{d_{h,k(h,x)}\} - \min_{J_{h,k(h,x)} \in \omega(V_x)} \{r_{h,k(h,x)}\} \geq \sum_{J_{h,k(h,x)} \in \omega(V_x)} C_{h,k(h,x)}, \quad \forall \omega(V_x) \subseteq \Omega(V_x). \tag{9}$$

Then for each pair of $r_{i,k(i,x)}$ and $d_{j,k(j,x)}$ with $J_{i,k(i,x)}, J_{j,k(j,x)} \in \Omega(V_x)$, there must exist some job subset $\omega(V_x)$ in $\Omega(V_x)$ such that

$$\omega(V_x) = \{J_{h,k(h,x)} | r_{h,k(h,x)} \geq r_{i,k(i,x)}, d_{h,k(h,x)} \leq d_{j,k(j,x)}\}.$$

Thus, $\forall J_{i,k(i,x)}, J_{j,k(j,x)} \in \Omega(V_x)$, we have

$$\max_{J_{h,k(h,x)} \in \omega(V_x)} \{d_{h,k(h,x)}\} - \min_{J_{h,k(h,x)} \in \omega(V_x)} \{r_{h,k(h,x)}\}$$

$$= d_{j,k(j,x)} - r_{i,k(i,x)} \geq \sum_{J_{h,k(h,x)} \in \omega(V_x)} C_{h,k(h,x)}$$

$$= \sum_{\substack{J_{h,k(h,x)} \in \Omega(V_x), \\ r_{h,k(h,x)} \geq r_{j,k(j,x)}, \\ d_{h,k(h,x)} \leq d_{i,k(i,x)}}} C_{h,k(h,x)}.$$

By Theorem 1, job set $\Omega(V_x)$ is schedulable.

Next, we prove the "only if" part. Assume that the job set $\Omega(V_x)$ is schedulable on V_x. According to Theorem 1, $\forall J_{i',k(i',x)}, J_{j',k(j',x)} \in \Omega(V_x)$, $r_{i',k(i',x)} \leq d_{j',k(j',x)}$, and

$$d_{j',k(j',x)} - r_{i',k(i',x)} \geq \sum_{\substack{J_{h,k(h,x)},\\ r_{h,k(h,x)} \geq r_{i',k(i',x)},\\ d_{h,k(h,x)} \leq d_{j',k(j',x)}}} C_{h,k(h,x)}. \quad (10)$$

Given any job subset $\omega(V_x)$, let $\max_{J_{i,k(i,x)} \in \omega(V_x)}\{d_{i,k(i,x)}\} = d_{j',k(j',x)}$ and $\min_{J_{i,k(i,x)} \in \omega(V_x)}\{r_{i,k(i,x)}\} = r_{i',k(i',x)}$. We have

$$\max_{J_{i,k(i,x)} \in \omega(V_x)}\{d_{i,k(i,x)}\} - \min_{J_{i,k(i,x)} \in \omega(V_x)}\{r_{i,k(i,x)}\}$$
$$\geq \sum_{\substack{J_{h,k(h,x)},\\ r_{h,k(h,x)} \geq r_{i',k(i',x)},\\ d_{h,k(h,x)} \leq d_{j',k(j',x)}}} C_{h,k(h,x)} \geq \sum_{J_{i,k(i,x)} \in \omega(V_x)} C_{i,k(i,x)}.$$

Therefore, constraint (6) holds true. □

Proof for Lemma 2: First, we prove that $r_{i,k(i,x)} + C_{i,k(i,x)} \leq d^*_{i,k(i,x)}$ for any $J_{i,k(i,x)}$ in any solution found by OLDA. Let $d^{\omega}_{i,k(i,x)} = \min_{J_{i,k(i,x)} \in \omega(V_x)}\{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)} \in \omega(V_x)} C_{i,k(i,x)}$ for any $\omega(V_x) \subseteq \Omega(V_x)$. According to Lines 4–14 of Algorithm 1, OLDA selects the job subset $\omega^*(V_x)$, which has the maximum $d^{\omega}_{i,k(i,x)}$ among all the subsets of $\Omega(V_x)$, and assigns this value to job $J_{*,k(*,x)}$ as its local deadline. That is,

$$d_{*,k(*,x)} = d^{\omega^*}_{i,k(i,x)} \geq d^{\omega}_{i,k(i,x)} \forall \omega(V_x) \subseteq \Omega(V_x). \quad (11)$$

Suppose that $d_{*,k(*,x)} < r_{*,k(*,x)} + C_{*,k(*,x)}$. In such a case, we can find a job subset $\omega'(V_x)$ containing $\{J_{*,k(*,x)}\}$ such that $d^{\omega'}_{i,k(i,x)} > d^{\omega^*}_{i,k(i,x)}$. This contradicts the condition in (11).

Next, we prove that $d^*_{i,k(i,x)} \leq D_i - \sum_{m=k(i,x)+1}^{M_i} C_{i,m}$ for any $J_{i,k(i,x)}$ in the solution found by OLDA. Suppose there is a job $J_{p,k(p,x)}$ where $d_{p,k(p,x)} > D_p - \sum_{m=k(p,x)+1}^{M_p} C_{p,m}$. In such a case, OLDA exits without returning a solution, which contradicts the assumption that OLDA returns a solution. Therefore, the solution returned by OLDA satisfies (4). □

Proof for Lemma 4: We prove the lemma by contradiction. Since $\omega^*(V_x)$ is the base subset of $\Omega(V_x)$, we have $\forall \omega(V_x) \subseteq \Omega(V_x)$,

$$r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)} \geq$$
$$\min_{J_{i,k(i,x)} \in \omega^(V_x)}\{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)} \in \omega(V_x)} C_{i,k(i,x)}. \quad (12)$$

Intuitively, the earliest start time of the first job in $\omega^*(V_x)$ is simply r^*, and the earliest completion time of the last job in $\omega^*(V_x)$ (denoted by f^*) satisfies $f^* = r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}$.

Suppose there are multiple idle time intervals inside $[r^*, f^*]$ when the jobs in $\omega^*(V_x)$ are executed. Let $T_{idle,sum}$ be the total duration of the idle times. Thus, the completion time of the last job in $\omega^*(V_x)$ can be expressed as

$$T_{complete,\omega^*(V_x)} = f^* + T_{idle,sum}. \quad (13)$$

Let $[t_{start}, t_{end}]$ be the latest idle time interval among all the idle time intervals. Under the work-conserving EDF policy, an idle interval means that no job is ready to be executed during the interval. In other words, t_{end} coincides with the release time $r_{q,k(q,x)}$ of some job $J_{q,k(q,x)}$. The completion time of $\omega^*(V_x)$ can also be expressed as,

$$T_{complete,\omega^*(V_x)} = r_{q,k(q,x)} + \sum_{\substack{J_{i,k(i,x)} \in \omega^*(V_x)\\ r_{i,k(i,x)} \geq r_{q,k(q,x)}}} C_{i,k(i,x)}. \quad (14)$$

This implies that,

$$r_{q,k(q,x)} + \sum_{\substack{J_{i,k(i,x)} \in \omega^*(V_x)\\ r_{i,k(i,x)} \geq r_{q,k(q,x)}}} C_{i,k(i,x)} > r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}.$$

This contradicts (12) stated earlier.

Since there is no idle time when executing the jobs in $\omega^*(V_x)$, and since the start time of the first job and the total execution time of the jobs in $\omega^*(V_x)$ are r^* and $\sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}$, respectively, the completion time of the last job in $\omega^*(V_x)$ is then equal to $r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}$. It follows that the busy interval is $[r^*, r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}]$. □

Proof for Lemma 5: Suppose there exists an optimal set of local deadlines $\{d^+_{i,k(i,x)}\}$ that is different from the solution $\{d^*_{i,k(i,x)}\}$ returned by OLDA. Then, there exists at least one job $J_{i,k(i,x)}$ whose $d^+_{i,k(i,x)}$ is different from $d^*_{i,k(i,x)}$. Our goal is to show that such differences in local deadline assignments do not affect the value of the objective function in (3).

Let the jobs in $\Omega(V_x)$ be examined in the order of the sequence in which a job obtains its local absolute deadline in OLDA. Suppose job $J_{p,k(p,x)}$ is the first job that has different deadlines $d^+_{p,k(p,x)}$ and $d^*_{p,k(p,x)}$ in solutions $\{d^+_{i,k(i,x)}\}$ and $\{d^*_{i,k(i,x)}\}$, respectively. According to Lemma 4, $d^*_{p,k(p,x)}$ is the completion time of the base job in base subset $\omega^*(V_x)$ and is equal to $r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}$. Hence, $d_{p,k(p,x)}$ is the longest absolute local deadline of the jobs in $\omega^*(V_x)$ for $\{d^*_{i,k(i,x)}\}$. Assume that in $\{d^+_{i,k(i,x)}\}$, the job $J_{q,k(q,x)}$ has the longest absolute local deadline among the jobs in $\omega^*(V_x)$. We consider the following scenarios: $p = q$ and $p \neq q$.

Case 1 $(p = q)$: Recall that $d_{p,k(p,x)} = r^* + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}$. It is trivial to verify that $d^+_{q,k(q,x)} \geq d^*_{p,k(p,x)}$. Hence, we have $S^*_{p,k(p,x)} \geq S^+_{q,k(q,x)}$, and the objective function of the OLDA's solution is larger than or equal to that of $J^+_{i,k(i,x)}$.

Case 2 $(p \neq q)$: In this case, we first prove that $S^+_{p,k} \geq S^+_{q,k}$ and $S^+_{p,k}$ does not influence the value of the objective function $\min_{\tau_{i,k(i,x)} \in \omega^+(V_x)}\{S^+_{i,k}\}$ directly. Next, we prove that $S^*_{p,k} \geq S^+_{q,k}$ and $S^*_{q,k} \geq S^+_{q,k}$, which leads to that the value of the objective function obtained by OLDA is larger than or equal to that of the assumed optimal solution.

Since $d^+_{q,k(q,x)} \geq d^+_{p,k(p,x)}$, and $D_p - \sum_{m=k(p,x)+1}^{M_p} C_{p,m} \geq D_q - \sum_{m=k(q,x)+1}^{M_q} C_{q,m}$ according to Line 16– 21 of Algorithm 1, we have $S^+_{p,k} \geq S^+_{q,k} \geq \min_{J_{i,k(i,x)} \in \omega^*(V_x)} \{S^+_{i,k}\}$. Therefore, $S^+_{p,k}$ does not influence the value of the objective function directly. In addition, since $d^+_{q,k(q,x)} \geq d^*_{p,k(p,x)}$, and $D_p - \sum_{m=k(p,x)+1}^{M_p} C_{p,m} \geq D_q - \sum_{m=k(q,x)+1}^{M_q} C_{q,m}$, we have $S^*_{p,k} \geq S^+_{q,k}$. Similarly, since $d^*_{q,k} \leq d^*_{p,k}$ and $d^*_{p,k} \leq d^+_{q,k}$, we have $S^*_{q,k} \geq S^+_{q,k}$. As a result, the value of the objective function obtained by OLDA is larger than or equal to that of the optimal solution in either case, hence the solution found by OLDA algorithm is optimal. □

Proof for Lemma 6: We prove Lemma 6 by contradiction. Suppose there is one solution $\{d_{i,k(i,x)}\}$ satisfying (4) and (6) but OLDA fails to find a solution. This means that the `if` condition on Line 22 in Algorithm 1 is true, and consequently, for job $J_{*,k(*,x)}$,

$$D_* - \sum_{m=k(*,x)+1}^{M_*} C_{*,m} < d_{*,k(*,x)}.$$

In addition,

$$d_{*,k(*,x)} = \min_{J_{i,k(i,x)} \in \omega^*(V_x)} \{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)}. \tag{15}$$

Given that $\Omega(V_x)$ is schedulable, according to Lemma 1, we have

$$\min_{J_{i,k(i,x)} \in \omega^*(V_x)} \{r_{i,k(i,x)}\} + \sum_{J_{i,k(i,x)} \in \omega^*(V_x)} C_{i,k(i,x)} \leq \max_{J_{i,k(i,x)} \in \omega^*(V_x)} \{d_{i,k(i,x)}\}. \tag{16}$$

Combining (15) and (16), we have

$$D_* - \sum_{m=k(*,x)+1}^{M_*} C_{*,m} < \max_{J_{i,k(i,x)} \in \omega^*(V_x)} \{d_{i,k(i,x)}\}. \tag{17}$$

Assume that $J_{q,k(q,x)}$ has the maximum local deadline among all the jobs in $\omega^*(V_x)$ in the feasible solution $\{d_{i,k(i,x)}\}$. Then,

$$D_* - \sum_{m=k(*,x)+1}^{M_*} C_{*,m} < \max_{J_{i,k(i,x)} \in \omega^*(V_x)} \{d_{i,k(i,x)}\} = d_{q,k(q,x)} \leq D_q - \sum_{m=k(q,x)+1}^{M_q} C_{q,m}. \tag{18}$$

In Algorithm 1, $J_{*,k(*,x)}$ is selected because it has the maximum upper bound on local deadline among all the jobs in $\omega^*(V_x)$ (Line 16– 21). However, $d_{q,k(q,x)} > d_{*,k(*,x)}$, which is a contradiction. □

References

[1] S. K. Baruah, L. E. Rosier, and R. R. Howell, “Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor,” *Real-Time Syst.*, vol. 2, no. 4, pp. 301–324, 1990.

[2] E. Bini and G. C. Buttazzo, “Biasing effects in schedulability measures,” in *ECRTS*, Jul. 2004, pp. 196–203.

[3] G. Buttazzo, E. Bini, and Y. Wu, “Partitioning parallel applications on multiprocessor reservations,” in *ECRTS*, 2010, pp. 24–33.

[4] S. Chatterjee and J. Strosnider, “Distributed pipeline scheduling: A framework for distributed, heterogeneous real-time system design,” 1995.

[5] H. Chetto, M. Silly, and T. Bouchentouf, “Dynamic scheduling of real-time tasks under precedence constraints,” *Real-Time Systems*, vol. 2, pp. 181–194, 1990.

[6] H. Chetto and M. Chetto, “Scheduling periodic and sporadic tasks in a real-time system,” *Information Processing Letters*, vol. 30, pp. 177–184, 1989.

[7] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, “Period optimization for hard real-time distributed automotive systems,” in *DAC*, 2007, pp. 278–283.

[8] W. Hawkins and T. Abdelzaher, “Towards feasible region calculus: An end-to-end schedulability analysis of real-time multistage execution,” in *RTSS*, 2005, pp. 75–86.

[9] P. Jayachandran and T. Abdelzaher, “Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling,” in *ECRTS*, 2008, pp. 233 –242.

[10] ——, “End-to-end delay analysis of distributed systems with cycles in the task graph,” in *ECRTS*, 2009, pp. 13 –22.

[11] ——, “Delay composition in preemptive and non-preemptive real-time pipelines,” *Real-Time Systems*, vol. 40, pp. 290–320, 2008.

[12] J. Jonsson and K. G. Shin, “Robust adaptive metrics for deadline assignment in distributed hard real-time systems,” *Real-Time Syst.*, vol. 23, pp. 239–271, November 2002.

[13] D. Marinca, P. Minet, and L. George, “Analysis of deadline assignment methods in distributed real-time systems,” *Computer Communications*, vol. 27, no. 15, pp. 1412 – 1423, 2004.

[14] S. Matic and T. Henzinger, “Trading end-to-end latency for composability,” in *RTSS*, 2005, pp. 12 pp. –110.

[15] J. Palencia and M. G. Harbour, “Offset-based response time analysis of distributed systems scheduled under edf,” *ECRTS*, pp. 3–12, 2003.

[16] A. Rahni, E. Grolleau, and M. Richard, “Feasibility Analysis of Non-Concrete Real-Time Transactions With EDF Assignment priority,” in *RTNS*, 2008.

[17] S. Samii, P. Eles, Z. Peng, and A. Cervin, “Quality-driven synthesis of embedded multi-mode control systems,” in *DAC*, July 2009, pp. 864 –869.

[18] N. Serreli, G. Lipari, and E. Bini, “The demand bound function interface of distributed sporadic pipelines of tasks scheduled by edf,” in *ECRTS*, 2010, pp. 187 –196.

[19] ——, “The distributed deadline synchronization protocol for real-time systems scheduled by edf,” in *ETFA*, 2010, pp. 1 –8.

[20] N. Serreli and E. Bini, “Deadline assignment for component-based analysis of real-time transactions,” in *WCRTS*, 2009.

[21] R. J. Vanderbei, “Loqo users manual c version 4.05,” 2006.

[22] Y. Zhang, R. West, and X. Qi, “A virtual deadline scheduler for window-constrained service guarantees,” in *RTSS*, 2004, pp. 151 – 160.

[23] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. S. Vincentelli, “Synthesis of task and message activation models in real-time distributed automotive systems,” in *DATE*, 2007, pp. 93–98.