Proceedings of the
46th IEEE Conference on Decision and Control
New Orleans, LA, USA, Dec. 12-14, 2007

Optimal Admission Control of Discrete Event Systems with Real-Time Constraints

Jianfeng Mao and **Christos G. Cassandras**
Dept. of Manufacturing Engineering
and Center for Information and Systems Engineering
Boston University
Brookline, MA 02446
jfmao@bu.edu, cgc@bu.edu

***Abstract*— The problem of optimally controlling the processing rate of tasks in Discrete Event Systems (DES) with hard real-time constraints has been solved in [9] under the assumption that a feasible solution exists. Since this may not always be the case, we introduce in this paper an admission control scheme in which some tasks are removed with the objective of maximizing the number of remaining tasks which are all guaranteed feasibility. In the off-line case where task information is known, we derive several optimality properties and develop a computationally efficient algorithm for solving the admission control problem under certain conditions. In the on-line case, we derive necessary and sufficient conditions under which idling is optimal and define a metric for evaluating when and how long it is optimal to idle. Numerical examples are included to illustrate our results.**

Keywords: **discrete event system, admission control, real-time constraints**

I. INTRODUCTION

A large class of Discrete Event Systems (DES) involves the control of resources allocated to tasks according to certain operating specifications (e.g., tasks may have real-time constraints associated with them). The basic modeling block for such DES is a single-server queueing system operating on a first-come-first-served basis, whose dynamics are given by the well-known max-plus equation

$$x_i = \max(x_{i-1}, a_i) + \mu_i \tau_i$$

where a_i is the arrival time of task $i = 1, 2, \ldots$, x_i is the time when task i completes service, μ_i is the number of operations and τ_i is its processing time per operation which is controllable (equivalently, the processing rate $1/\tau_i$ is controllable). Examples arise in manufacturing systems, where the operating speed of a machine can be controlled to trade off between energy costs and requirements on timely job completion [11]; in computer systems, where the CPU speed can be controlled to ensure that certain tasks meet specified execution deadlines [2],[6]; and in wireless networks where severe battery limitations call for new techniques aimed at maximizing the lifetime of such a network [3],[10]. A particularly interesting class of problems arises when such systems are subject to *real-time constraints*, i.e., $x_i \leq d_i$ for each task i with a given "deadline" d_i. In order to meet such constraints, one typically has to incur a higher cost associated with control τ_i. Thus, in a broader context, we are interested in studying optimization problems of the form:

$$\begin{aligned} &\min_{\tau_1,\ldots,\tau_N} \left\{ \sum_{i=1}^{N} \mu_i \theta(\tau_i) \right\} \qquad (1) \\ \text{s.t.} \quad &x_i = \max(x_{i-1}, a_i) + \mu_i \tau_i \leq d_i, \qquad i = 1, ..., N; \\ &\tau_i \geq \tau_{\min}, \qquad i = 1, ..., N. \end{aligned}$$

where $\theta(\tau_i)$ is a given cost function, $\tau_{\min}$ is the minimal processing time per operation, and all a_i, d_i are known. Such problems have been studied for preemptive tasks [12],[1], nonpreemptive periodic tasks [4],[5], and nonpreemptive aperiodic tasks [3],[10],[9]. The latter case is of particular interest in wireless communications where nonpreemptive scheduling is necessary to execute aperiodic packet transmission tasks which also happen to be highly energy-intensive; in such cases, the cost function in (1) represents the energy required for a packet transmission. One of the key challenges in dealing with (1) is to develop computationally efficient solution approaches that can be used in real-time settings and can be implemented in wireless devices with very limited computational power.

In prior work [9], we have shown that exploiting structural properties of the optimal state trajectory in (1) leads to a highly efficient *Critical Task Decomposition Algorithm* (CTDA) for obtaining a solution *as long as the problem is feasible*. This leaves open the question of dealing with the case where feasibility does not hold, which may often arise in practice. For example, tasks may arrive in a bursty fashion causing a temporary system overload. When this happens, some tasks will violate their real-time constraints even if all tasks are processed with the minimal processing time per operation. Thus, before the CTDA can be applied it is necessary to selectively reject tasks so as to render the problem feasible. This naturally leads to an *admission control* problem where the objective becomes to maximize the number of tasks allowed to remain in the system while guaranteeing the feasibility of (1). The contribution of this paper is to formulate this problem and develop an efficient

The authors' work is supported in part by the National Science Foundation under Grant DMI-0330171, by AFOSR under grants FA9550-04-1-0133 and FA9550-04-1-0208, and by DOE under grant DE-FG52-06NA27490.

1-4244-1498-9/07/$25.00 ©2007 IEEE.

solution we term *Maximal Shift Task Removal Algorithm* (MSTRA) under certain sufficient conditions. Further, we consider the on-line version of this problem where task arrival information is not known in advance and show that under certain necessary and sufficient conditions *idling is optimal* (in the sense of maximizing the number of tasks that can be processed without violating their deadlines). We propose a metric based on distributional information characterizing task arrivals and develop an algorithm for solving the on-line version of (1).

In Section II of the paper we formulate the admission control problem associated with (1). In Section III, we exploit some optimality properties leading to the aforementioned MSTRA. We consider the on-line admission control problem in Section IV, provide numerical examples in Section V, and conclude with Section VI.

II. Problem Formulation

In this section, we will formulate the admission control problem whose solution will maximize the number of tasks leading to a feasible optimization problem (1). The first step is to develop a necessary and sufficient condition for easily checking the feasibility of (1). Let $\hat{x}_i$ denote the departure time of task i when all tasks are processed in $\tau_{\min}$, i.e., the system dynamics reduce to

$$\hat{x}_i = \max(\hat{x}_{i-1}, a_i) + \mu_i \tau_{\min}$$

Based on the lemma below, we can easily determine the feasibility of (1) by checking whether $\hat{x}_i \leq d_i$ for all $i = 1, ..., N$.

Lemma 1: Problem (1) is feasible if and only if $\hat{x}_i \leq d_i$ for all $i = 1, ..., N$.

(Most of the proofs in this paper are omitted or just sketched due to space limitations; the full proofs can be found in [7].)

By Lemma 1, the feasibility of (1) is equivalent to the feasibility of the solution $\tau_i = \tau_{\min}$ for $i = 1, ..., N$. Based on this fact, we formulate the **Admission Control Problem** (ACP) as follows:

$$\max_{z_1,...,z_N} \sum_{i=1}^{N} z_i$$
$$\text{s.t. } x_i = \max(x_{i-1}, a_i z_i) + \mu_i \tau_{\min} z_i,\ i = 1, ..., N;$$
$$(x_i - d_i) z_i \leq 0,\ z_i \in \{0, 1\},\ i = 1, ..., N;\ x_0 = 0.$$

where $0 \leq a_1 \leq a_2 \leq ... \leq a_N$ and $z_i \in \{0, 1\}$. Thus, $z_i = 0$ means that the i-th task is removed, while $z_i = 1$ implies it is admitted. Note that the objective function is simply the number of admitted tasks. The constraints are different from (1): If $z_i = 1$, then task i must follow the same dynamics as in (1) and it has to meet the hard real time constraint $x_i \leq d_i$; if $z_i = 0$, then the departure time of task i, x_i, is assigned to coincide with x_{i-1} and the associated hard real time constraint is overlooked. The ACP is an integer programming problem with nonlinear inequality constraints. Although it may be solved by standard methods (e.g., branch and bound), this is too time consuming to be implemented in a real-time environment with limited resources. In what follows, we develop an efficient solution algorithm by utilizing the optimality properties presented in the next section.

III. Optimality Properties

Before we formally study the optimality properties of the ACP, we consider a simple greedy algorithm, termed *Direct Deletion Algorithm* (DDA). The idea is to process all tasks in $\tau_{\min}$ and drop the current task when it violates its deadline. The algorithm has very low complexity, but it is not optimal as the following counterexample shows: $a_1 = \cdots = a_5 = 0$, $d_1 = \cdots = d_5 = 9$, $\mu_1 = 8$, $\mu_2 = \cdots = \mu_5 = 2$ and $\tau_{\min} = 1$. If we apply the DDA, tasks $\{2, ..., 5\}$ will be dropped. However, the obvious optimal solution is to remove only task 1 so that the remaining tasks $\{2, ..., 5\}$ can meet their deadlines.

Although the DDA fails to be optimal, it provides a clue regarding optimality. In the counterexample above, the failure of the method is caused by focusing only on the task that violates its deadline. Instead, we should consider all tasks before the one violating its deadline and pick an appropriate one to drop, which may lead to an optimal solution. Intuitively, the "appropriate" task should be the one that results in the largest departure time shift of the remaining tasks. In what follows, we will explore this idea further by first defining the concepts of "busy period" and "first infeasible task".

A. Busy Period and First Infeasible Task

Definition 1: A *Busy Period* (BP) is a set of contiguous tasks $\{k, ..., n\}$, such that $\hat{x}_{k-1} \leq a_k$, $\hat{x}_n \leq a_{n+1}$ and $\hat{x}_i > a_{i+1}$ for $i = k, ..., n-1$.

Based on the definition of a BP, we can decompose the ACP into a set of smaller problems, one for each BP $\{k, ..., n\}$, that is,

$$\max_{z_k,...,z_n} \sum_{i=k}^{n} z_i$$
$$\text{s.t. } x_k = a_k + \mu_k \tau_{\min} z_k;$$
$$x_i = \max(x_{i-1}, a_i z_i) + \mu_i \tau_{\min} z_i,\ i = k+1, ..., n;$$
$$(x_i - d_i) z_i \leq 0, \quad z_i \in \{0, 1\}, \quad i = k, ..., n.$$

Since the removal of some task can only decrease the departure time of all following tasks, any interdependence among BPs can only result from the tasks before a specific BP. Without loss of generality, we remove some task before task k. This will result in a new departure time x_{k-1} which must be no larger than $\hat{x}_{k-1}$. From the property $\hat{x}_{k-1} \leq a_k$ of a BP, the removed task cannot affect those tasks in the BP $\{k, ..., n\}$. Therefore, the optimal solution of the ACP can be obtained by independently solving these smaller problems. Based on the definition of a BP, we define next its first infeasible task.

Definition 2: Suppose task m belongs to the BP $\{k, ..., n\}$. If $x_m > d_m$ and $x_i \leq d_i$ for $i = k, ..., m-1$, then task m is the *first infeasible task*.

The following lemma identifies a property of the first infeasible task.

Lemma 2: Suppose task m is the first infeasible task in the BP $\{k, ..., n\}$. Then, for any feasible solution $[z_1, ..., z_N]$ of the ACP, there must exist some task j ($k \leq j \leq m$) such that $z_j = 0$.

B. Maximal Shift Task

In this section, we focus on a BP $\{k, ..., n\}$ which has a first infeasible task m. We first define the departure time shift of a task i resulting from the removal of *only* task j such that $k \leq j \leq i \leq m$. We denote this shift by $v_i(j)$.

Definition 3: Let

$$c_i(j) = \min_{s=j,...,i-1} \{\hat{x}_s - a_{s+1}\} \tag{2}$$

The shift of task i when only task j is removed is

$$v_i(j) = \min\left(c_i(j), \mu_j \tau_{\min}\right) \tag{3}$$

for any i, j such that $k \leq j \leq i \leq m$.

Note that $\hat{x}_s - a_{s+1}$ represents the waiting time of task $s+1$ within the BP $\{k, ..., n\}$. The smallest such waiting time is the extent to which removing a task j can cause a departure time shift in a subsequent task unless the processing time of j, $\mu_j \tau_{\min}$, is even smaller.

Lemma 3: For any $s \in \{j, \ldots, i\}$,

$$v_i(j) = \min\left(c_i(s), v_s(j)\right)$$

Lemma 4: Let $x_i(j)$ denote the departure time of task i after removing only task j. Then, $v_i(j) = \hat{x}_i - x_i(j)$ for any i, j such that $k \leq j \leq i \leq m$.

Definition 4: Task r is the maximal shift task if

$$v_m(r) \geq v_m(j), \quad \forall\, j = k, ..., r-1. \tag{4}$$

$$v_m(r) > v_m(j), \quad \forall\, j = r+1, ..., m; \tag{5}$$

The theorem below identifies an important optimality property of the maximal shift task defined above.

Theorem 1: Suppose m is the first infeasible task in the BP $\{k, ..., n\}$ and r is the maximal shift task. If $r = m$ or $v_m(r) \geq \hat{x}_m - d_m$, then there must exist an optimal solution $[z_1^*, ..., z_N^*]$ of the ACP such that $z_r^* = 0$.

Proof: [sketch] Assume on the contrary that $z_r^* = 1$ in all ACP solutions. From Lemma 2, there always exists some task q ($k \leq q \leq m$) removed in any feasible solution. Without loss of generality, there must exist an optimal solution $[z_1^*, ..., z_N^*]$ such that $z_r^* = 1$ and $z_q^* = 0$ ($q \neq r$).

Let us construct a potential solution $[\bar{z}_1, ..., \bar{z}_N]$ such that $\bar{z}_r = 0$, $\bar{z}_q = 1$ and $\bar{z}_i = z_i^*$ for all $i \neq q, r$. If we can show that $[\bar{z}_1, ..., \bar{z}_N]$ is a feasible solution, then $[\bar{z}_1, ..., \bar{z}_N]$ must also be optimal because $\sum_{i=1}^N \bar{z}_i = \sum_{i=1}^N z_i^*$, which contradicts the assumption that r is admitted in all optimal solutions. Therefore, the theorem can be proved by showing the feasibility of the potential solution $[\bar{z}_1, ..., \bar{z}_N]$.

Let x_i^* and $\bar{x}_i$ denote the departure time of task i resulting from the optimal solution $[z_1^*, ..., z_N^*]$ and the potential solution $[\bar{z}_1, ..., \bar{z}_N]$ respectively. In the following, we will prove the feasibility of $[\bar{z}_1, ..., \bar{z}_N]$, that is,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = 1, ..., N \tag{6}$$

First, we prove (6) for $i = 1, ..., k-1$, i.e.,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = 1, ..., k-1 \tag{7}$$

Since $[z_1^*, ..., z_N^*]$ must be feasible, we have

$$z_i^*(x_i^* - d_i) \leq 0, \quad \forall\, i = 1, ..., N \tag{8}$$

Since $\bar{z}_i = z_i^*$ for $i = 1, ..., k-1$, we have $\bar{x}_i = x_i^*$ for $i = 1, ..., k-1$. Combining this with (8), we obtain (7).

Second, we prove (6) for $i = k, ..., m-1$, i.e.,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = k, ..., m-1 \tag{9}$$

Since m is the first infeasible task, we have

$$\hat{x}_i \leq d_i \quad \forall\, i = k, ..., m-1 \tag{10}$$

Since $\bar{x}_i$ is the departure time after the possible removal of some tasks,

$$\bar{x}_i \leq \hat{x}_i, \quad \forall\, i = 1, ..., N \tag{11}$$

Combining (11) and (10), we have $\bar{x}_i \leq d_i$ for all $i = k, ..., m-1$, which implies (9).

Third, we prove (6) for $i = m$, i.e.,

$$\bar{z}_m(\bar{x}_m - d_m) \leq 0 \tag{12}$$

This follows from the assumption that either $r = m$ or $v_m(r) \geq \hat{x}_m - d_m$. If $r = m$, then $\bar{z}_m = \bar{z}_r = 0$ so that (12) immediately follows. If $v_m(r) \geq \hat{x}_m - d_m$, it follows from Lemma 4 that $x_m(r) \leq d_m$. Since at least task r is removed in $[\bar{z}_1, ..., \bar{z}_N]$, we have $\bar{x}_m \leq x_m(r)$. Therefore, it immediately follows that $\bar{x}_m \leq d_m$, which also implies (12).

Finally, we prove (6) for $i = m+1, ..., N$, i.e.,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = m+1, ..., N \tag{13}$$

for which there are two possible cases.

Case 1: No task between k and r is removed in the optimal solution $[z_1^*, ..., z_N^*]$, that is, $z_i^* = 1$ for all $i = k, ..., r$. In this case, we select q to be the first task removed after r, that is, $q > r$, $z_q^* = 0$, and $z_i^* = 1$ for $i = r, ..., q-1$. We can obtain (6) from Lemma 3 and Lemma 4.

Case 2: There exists some task between k and r which is removed in $[z_1^*, ..., z_N^*]$, that is, there exists some task i ($k \leq i < r$) such that $z_i^* = 0$. In this case, we select q to be the last task removed before r, that is, $q < r$, $z_q^* = 0$, and $z_i^* = 1$ for $i = q+1, ..., r$. We can once again establish (13), which completes the proof. ■

The following Corollary of Theorem 1 establishes the fact that the conditions under which the theorem holds are satisfied for a large class of problems.

Corollary 1: If $d_1 \leq d_2 \leq ... \leq d_N$, then there must exist an optimal solution $[z_1^*, ..., z_N^*]$ of the ACP such that $z_r^* = 0$.

Theorem 1, leads directly to a highly efficient admission control algorithm we term *Maximal Shift Task Removal Algorithm* (MSTRA) shown in Table I. The algorithm yields the optimal solution of the ACP when the condition $r = m$ or $v_m(r) \geq \hat{x}_m - d_m$ is satisfied for all BPs (e.g., in the case in Corollary 1). Otherwise, the algorithm can still obtain a near-optimal solution. The difference between the performance of the MSTRA and the optimal performance is bounded by

the number of times that the condition is violated over the whole process. Regarding the complexity of the MSTRA, since we can locate the maximal shift task in $O(N)$, the total complexity is $O(N^2)$.

TABLE I
MAXIMAL SHIFT TASK REMOVAL ALGORITHM

Step 1:	Locate a busy period $\{k, ..., n\}$ by comparing $\hat{x}_i$ and a_{i+1} and identify its first infeasible task m;
Step 2:	Compute $v_m(i)$ for $i = k, ..., m$ from (2) (3) and find the maximal shift task r from (4) (5);
Step 3:	If $r = m$ or $v_m(r) \geq \hat{x}_m - d_m$, then remove task r; otherwise remove task m.
Step 4:	If the problem with the remaining tasks is feasible, then end. Otherwise, goto Step 1.

IV. ON-LINE ADMISSION CONTROL

So far, we have assumed that all task arrivals, number of operations, and deadlines are known at the time we solve the ACP and, subsequently, the original problem (1). This corresponds to an *off-line* approach. In this section, we tackle the *on-line* admission control problem where this information is unavailable until a task actually arrives. Therefore, one can proceed in two ways: (i) Based only on the information available for tasks already in queue, and (ii) Using information in the form of the joint distribution of the arrival time, deadline, and required number of operations of the next arriving task.

We begin with case (i). Without loss of generality, we assume the current decision point is at time $t > 0$. Note that all tasks that arrived before t and have not been processed yet can be regarded as a set of tasks sharing a common arrival time t. It is then possible to show that the optimal way to process these tasks is using the *Earliest Deadline First* (EDF) policy as established in the next lemma.

Lemma 5: If tasks share a common arrival time, then EDF is the processing policy that maximizes the number of remaining tasks that can be processed without violating their deadlines.

Using this lemma, we can sort tasks in ascending order of deadlines and apply the MSRTA to obtain the optimal admission control for all tasks in queue from Corollary 1.

Next, we consider case (ii), i.e., making decisions based on uncertain information about the next arrival task. Two natural questions arise: Is it possible to improve performance by using such information, and, if so, how to accomplish it? The first question can be answered through the following simple example. Imagine that the next arrival task is imminent and has an urgent deadline, while the current task has a large number of operations and a relatively loose deadline. Then, it is very likely that the next task will be dropped when it arrives while we still process the current task. If distribution information for the next arrival task is available, it is possible to compute some metric quantifying the likelihood that the case above can occur. If the metric shows it can happen with sufficiently high probability, then we may rescue the upcoming task by postponing the start of the current task (i.e., by explicitly *idling*) and still successfully process the postponed task. The net effect is better performing admission control with high probability.

To answer the second question, we can get some clues from the answer to the first one. In particular, the key is to specify the aforementioned metric and compute it. The theorem below is a step in this direction. Without loss of generality, assume that at the current decision time, t, there are n remaining tasks after applying admission control through the MSTRA and they satisfy $d_1 \leq d_2 \leq ... \leq d_n$ and $t + \sum_{j=1}^{i} \mu_j \tau_{\min} \leq d_i$ for $i = 1, ..., n$ (as already mentioned, the MSTRA yields the optimal solution in this case). Moreover, the optimal processing time per operation can be assigned to task 1 using the CTDA [9] which solves problem (1). Let τ_1 be this processing time and let a_{n+1}, d_{n+1} and μ_{n+1} denote the arrival time, deadline, and number of operations of the next arrival task respectively. We are interested in establishing necessary and sufficient conditions under which the optimal number of remaining tasks that are all feasible (determined through the MSTRA *without any future information available*) can be improved by idling *given some future information*. Then, clearly the optimal number of remaining tasks that are all feasible is at least n. If we consider the next arriving task only, then the best we can do is increase this number to $n + 1$.

It follows that the statement "we can improve the optimal number of remaining tasks that are feasible under no future information" is equivalent to the two conditions:

Condition 1. The optimal number of remaining tasks is n if we do not postpone task 1.

Condition 2. The optimal number of remaining tasks is $n + 1$ if we idle and postpone task 1.

Theorem 2: Assume the current decision point is t and there are n tasks in queue that are feasible. Let $\Phi(i)$ denote the set of tasks before task i and task i itself when all $n+1$ tasks are sorted in ascending order of deadline and $\Psi(i) = \Phi(i) \backslash \{1\}$. *Condition 1* and *Condition 2* are satisfied if and only if

$$\exists\, i \in \{2, ..., n+1\},\ \ t + \mu_1\tau_1 + \sum\nolimits_{j \in \Psi(i)} \mu_j \tau_{\min} > d_i; \quad (14)$$

$$a_{n+1} + \sum\nolimits_{j \in \Phi(i)} \mu_j \tau_{\min} \leq d_i,\ \forall\, i = 1, ..., n+1. \quad (15)$$

Proof: $\Longrightarrow$: First, we prove that

$$a_{n+1} \leq t + \mu_1 \tau_1, \quad (16)$$

Using the definitions of $\Phi(i)$ and $\Psi(i)$, we have

$$\sum\nolimits_{j \in \Psi(i)} \mu_j \tau_{\min} \leq \sum\nolimits_{j \in \Phi(i)} \mu_j \tau_{\min} \quad (17)$$

By (14), there exists some task i such that

$$t + \mu_1 \tau_1 + \sum\nolimits_{j \in \Psi(i)} \mu_j \tau_{\min} > d_i$$

By (15), we have

$$a_{n+1} + \sum\nolimits_{j \in \Phi(i)} \mu_j \tau_{\min} \leq d_i$$

Combining the three inequalities above, we obtain the inequality (16).

Second, we consider the case where task 1 is not postponed, implying that $t + \mu_1\tau_1$ is the next decision point. From (16), task $n+1$ has arrived at this decision point so that $t + \mu_1\tau_1 + \sum_{j\in\Psi(i)} \mu_{(j)}\tau_{\min}$ is the earliest departure time of task i for all $i = 2, ..., n+1$.

Based on Lemma 1, we can check feasibility by analyzing the earliest departure time which is achieved through the minimal processing time per operation $\tau_{\min}$. Then, from (14), at least one of tasks $2, ..., n+1$ will violate its deadline. By assumption, there are n feasible tasks in queue so that we can always attain the feasibility of these n tasks by removing task $n+1$. Therefore, the optimal number of remaining tasks is n if we do not postpone task i, i.e., *Condition 1* is met.

Next, we consider postponing task 1, implying that a_{n+1} is the next decision point and $a_{n+1} + \sum_{j\in\Phi(i)} \mu_{(j)}\tau_{\min}$ is the earliest departure time of task i for all $i = 1, ..., n+1$.

Similarly, we can still check feasibility based on the earliest departure time from Lemma 1. Then, from (15), all $n+1$ tasks can meet their deadlines. Thus, *Condition 2* is satisfied as well.

$\Longleftarrow$: First, we consider postponing task 1. From *Condition 2* and Lemma 1, all $n+1$ tasks can meet their deadlines by using the minimal processing time per operation $\tau_{\min}$, which turns out to be the inequality (15).

Second, we prove (16) has to be satisfied. Assume on the contrary that $a_{n+1} > t + \mu_1\tau_1$. From (17) and (15) obtained above, we have

$$a_{n+1} + \sum\nolimits_{j\in\Psi(i)} \mu_j\tau_{\min} \le d_i, \quad \forall\, i = 2, ..., n+1$$

Combining it with $a_{n+1} > t + \mu_1\tau_1$, all the other tasks can still meet their deadline by using $\tau_{\min}$ if we do not postpone task 1, which means the optimal number of remaining task also equals to $n+1$ in the case of not postponing task 1. This contradicts *Condition 1* and leads to the conclusion that (16) must hold.

Finally, we consider the case where task 1 is not postponed. It follows from (16) that $t+\mu_1\tau_1+\sum_{j\in\Psi(i)} \mu_{(j)}\tau_{\min}$ is the earliest departure time of task i for all $i = 2, ..., n+1$. Then, from *Condition 1* and Lemma 1, if we do not postpone task 1, there is at least one among tasks $2, ..., n+1$ such that its earliest departure time is larger than its deadline, which is precisely inequality (14). ■

Let f denote the joint distribution of $(a_{n+1}, d_{n+1}, \mu_{n+1})$ and $P(t, f)$ denote the probability that $(a_{n+1}, d_{n+1}, \mu_{n+1})$ satisfies (14) and (15) at the decision point t. From Theorem 2, $P(t, f)$ can be the metric to indicate how likely idling and postponing the current task could improve performance. Thus, if $P(t, f) > p$ (typically, $p = 0.5$), then we postpone the current task and otherwise immediately process it. Although $P(t, f)$ may not be easy to compute in closed form, one can always estimate it through Monte Carlo methods, that is, randomly generating M samples of $(a^i_{n+1}, d^i_{n+1}, \mu^i_{n+1})$ and calculating $\sum_{i=1}^{M} \mathbf{1}(a^i_{n+1}, d^i_{n+1}, \mu^i_{n+1})/M$, where $\mathbf{1}(\cdot)$ is the indicator function indicating whether $(a^i_{n+1}, d^i_{n+1}, \mu^i_{n+1})$ satisfies (14) and (15).

Further, if the decision is to idle, there remains a question regarding the length of idling. Let $P(t + w, f)$ denote the probability that $(a_{n+1}, d_{n+1}, \mu_{n+1})$ satisfies (14) and (15) if we postpone the current task by w and the next task still does not arrive. Define

$$\Omega = \{w : P(t + w, f) \le 0.5,\ w \ge 0\} \text{ and } \ w^* = \min_{w\in\Omega} w.$$

We can see that w^* is the optimal idling time. However, w^* is much harder to compute in closed form than $P(t, f)$. Let $L = \min_{i=1,...,n}(d_i - \sum_{j=1}^{i} \mu_i\tau_{\min})$. We can bound w^* as follows: $0 \le w^* \le L - t$, because (15) requires the next task to arrive before L. Moreover, it is clear that the longer idling is, the less likely we are to increase the optimal number of remaining tasks since less time becomes available for rescuing the next task. Thus, assuming that $P(t + w, f)$ is monotonically decreasing in w, an estimate of w^* may be efficiently obtained through simple binary search over the interval $[0, L - t]$. This discussion leads to the on-line admission control algorithm shown in Table II.

TABLE II
ON-LINE ADMISSION CONTROL ALGORITHM

Step 1:	Sort all tasks waiting in queue in ascending order of deadline and apply MSTRA and CTDA;
Step 2:	Estimate $P(t, f)$ by Monte Carlo method;
Step 3:	If $P(t, f) > 0.5$, then estimate w^* and postpone the current task for w^*; otherwise just process the current task.

V. NUMERICAL RESULTS

A. Off-line control

We first compare the performance and the complexity of two off-line algorithms: the MSTRA and the DDA. Both are implemented using Matlab 7.0 on an Intel(R) Core(TM)2 CPU 1.86 GHz, 1.0 GB RAM Computer. We test cases where N varies from 100 to 1000 in increments of 100. We randomly generated 50 samples for each N, in which task interarrival times are exponentially distributed with mean 8, μ_i are selected from 1 to 10 with equal probability and $d_i - a_i$ are uniformly distributed over $[2\mu_i\tau_{\min}, 2\mu_i(\tau_{\min} + 1)]$.

Figure 1 compares the performance of these two algorithms, in which the y-axis is the average number of tasks removed. We see that the MSTRA can remove about $13 \sim 16\%$ fewer tasks to attain feasibility in (1) than the DDA. Figure 2 shows the complexities of these two algorithms in average CPU time. Although the DDA has a lower complexity, the MSTRA is also very efficient (e.g., it can solve the case with $N = 1000$ in less than 0.1 seconds).

B. On-line control

Next, we compare two on-line admission control algorithms: one without idling and the other with idling. The setting of the on-line experiments is the same as above. Figure 3 compares the performance of these two on-line

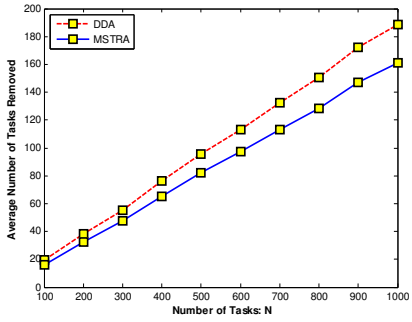

Fig. 1. Performance comparison of two off-line algorithms

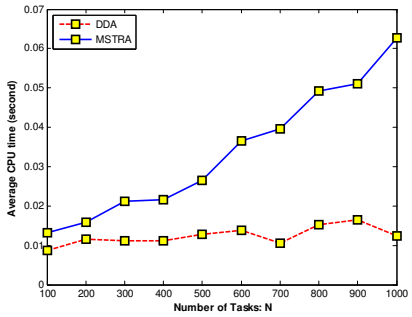

Fig. 2. Complexity comparison of two off-line algorithms

algorithms, in which the y-axis is the average number of tasks removed. We see that idling results in removing about $28 \sim 31\%$ fewer tasks compared to no idling. Figure 4 compares the complexity of these two on-line algorithms in terms of average CPU time. Obviously, on-line admission control without idling tasks has a lower complexity because it does not need to calculate the metric $P(t, f)$ and the length of idling. However, the algorithm with idling tasks is still very efficient (e.g., it can solve the case with $N = 1000$ in less than 0.9 seconds).

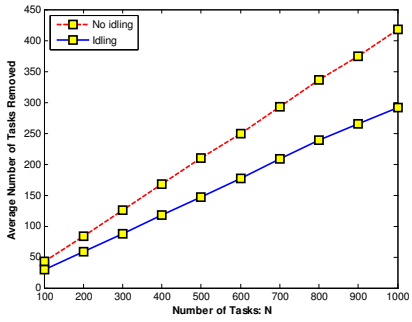

Fig. 3. Performance comparison of two on-line algorithms

VI. Conclusions

In this paper, we have removed the assumption that a feasible solution exists in Problem (1) and formulated an associated admission control problem for maximizing the number of tasks processed which are guaranteed feasibility. For the off-line admission control problem, we exploit an optimality property based on removing the "maximal shift task"

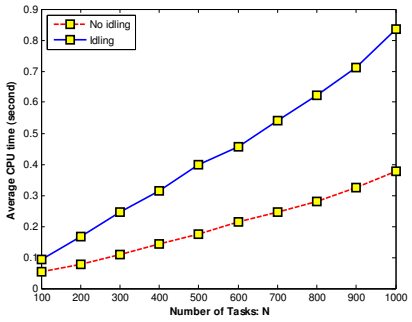

Fig. 4. Complexity comparison of two on-line algorithms

and develop an efficient admission control algorithm, termed Maximal Shift Task Removal Algorithm (MSTRA) which can obtain an optimal solution in most cases. We further develop an on-line admission control algorithm by utilizing the MSTRA and a metric based on distributional information for the next arrival task which is used to determine whether it is beneficial to idle and for how long.

Problem (1) characterizes a single-stage DES. Multi-stage systems studied in [8] face similar feasibility issues, which are complicated by the coupling among stages. Our ongoing work is aimed at studying the admission control problem in such systems.

References

[1] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Computers*, 53(5):584 – 600, May 2004.

[2] G.C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, 1997.

[3] A. E. Gamal, C. Nair, B. Prabhakar, Elif Uysal-Biyikoglu, and S. Zahedi. Energy-efficient scheduling of packet transmissions over wireless networks. In *Proceedings of IEEE INFOCOM*, volume 3, 23-27, pages 1773–1782, New York City, USA, 2002.

[4] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 129–139, 1991.

[5] J. Jonsson, H. Lonn, and K.G. Shin. Non-preemptive scheduling of real-time threads on multi-level-context architectures. In *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, volume 1586, pages 363–374. Springer Verlag, 1999.

[6] J.W.S Liu. *Real - Time System*. Prentice Hall Inc., 2000.

[7] J. Mao and C.G. Cassandras. Optimal admission control of discrete event systems with real-time constraints. *Technical Report, CODES, Boston University*, 2007. See also ftp://dacta.bu.edu:2491/TechReport/2007MSTRA.pdf.

[8] J. Mao and C.G. Cassandras. Optimal control of multi-stage discrete event systems with real-time constraints. In *Proc. of 45rd IEEE Conf. Decision and Control*, pages 1057–1062, Dec. 2006. (subm. to IEEE Trans. on Automatic Control, 2007).

[9] J. Mao, C.G. Cassandras, and Q.C. Zhao. Optimal dynamic voltage scaling in power-limited systems with real-time constraints. *IEEE Trans. on Mobile Computing*, 6(6):678–688, June 2007.

[10] L. Miao and C. G. Cassandras. Optimal transmission scheduling for energy-efficient wireless networks. In *Proceedings of INFOCOM*, 2006.

[11] D.L. Pepyne and C.G. Cassandras. Optimal control of hybrid systems in manufacturing. In *Proceedings of the IEEE*, volume 88, pages 1108–1123, 2000.

[12] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 374–382. IEEE Computer Society, 1995.