

# On controlling prioritized discrete event systems with real-time constraints

Lei Miao<sup>1</sup> 

Received: 14 July 2017 / Accepted: 12 April 2018 / Published online: 26 April 2018  
© Springer Science+Business Media, LLC, part of Springer Nature 2018

**Abstract** We study a class of prioritized Discrete Event Systems (DESs) that involve the control of resources allocated to tasks under real-time constraints. Our work is motivated by applications in communication systems, computing systems, and manufacturing systems where the objective is to minimize energy consumption while guaranteeing that task deadlines are always met. In the off-line setting, we discover several structural properties of the optimal sample path of such DESs. Using the structural properties, we also propose a greedy algorithm which is shown numerically near optimal. For on-line control, we design a Receding Horizon (RH) controller. Using worst-case estimation, the RH control is able to guarantee feasibility (when the off-line problem is feasible) and achieve good performance.

**Keywords** Optimization · Energy-efficiency · Real-time systems · Receding horizon control · Discrete event systems

## 1 Introduction

A number of Discrete Event Systems (DESs) with real-time constraints have been studied recently. This is motivated by applications in power-limited systems where a trade-off between resource efficiency and system performance exists. Examples include Dynamic Voltage Scaling (Mao et al. 2007) and Dynamic Transmission Scheduling (Uysal-Biyikoglu et al. 2002; Miao and Cassandras 2017). In those motivating applications, the system cost, i.e., energy, is highly related to the speed at which the system is operating at. The goal is to conserve energy by adjusting the control (processing/transmission rate) and also maintain satisfactory system performance in terms of latency. The basic modeling block for such

---

✉ Lei Miao  
lei.miao@mts.u.edu

<sup>1</sup> Mechatronics Engineering, 1301 East Main Street, Box 19, Murfreesboro, TN 37132-0001, USA

DESS is a single-server queueing system operating on a first-come-first-served (FCFS) basis, whose dynamics are given by the well-known max-plus equation

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

where  $a_i$  is the arrival time of task  $i = 1, 2, \dots$ ,  $x_i$  is the time when task  $i$  completes serve, and  $s_i$  is its (generally random) service time. Under the system dynamics specified in Eq. (1) and a real-time constraint that must be met for each task, the objective of the control and optimization problem is to minimize the total energy consumed over some given  $N$  tasks. Note that although our model is similar to that of a single-server queueing system, there are differences between the DESSs we study in this paper and queueing systems: (i) In our analysis, we do not rely on stochastic models, which are often used by traditional queueing theoretic techniques and (ii) Our main goal is to perform optimal control whereas in queueing theory, the main goal is often to determine the system's performance under certain operating conditions.

The optimization problem in the single-node case has been studied in Mao et al. (2007), and it turns out that although the constraints contain nondifferentiable max-plus equations, the problem can be solved efficiently using the Critical Task Decomposition Algorithm (CTDA) (Mao et al. 2007), as long as the cost function is strictly convex, differentiable, and monotonically increasing (or decreasing, depending on the control variable). CTDA is very suitable for power-limited devices because it only requires solving a set of simple linear equations. Note that CTDA only works when the optimization problem is feasible. When it is infeasible, an admission control problem is studied in Mao and Cassandras (2010), in which certain tasks are dropped in order to maximize the number of remaining tasks that have guaranteed real-time constraints.

While CTDA is developed for a single-node case, other work has been done in a network setting where multiple nodes are involved. In Gamal et al. (2002) and Miao and Cassandras (2017), a Downlink Transmission Scheduling problem is formulated and studied. DTS involves a single transmitter and multiple receivers, and the cost function at the transmitter is *task-dependent*. Using the Generalized Critical Task Decomposition Algorithm (GCTDA) algorithm developed in Miao and Cassandras (2017), the original complex nonlinear optimization problem in DTS boils down to solving a sequence of nonlinear algebraic equations. In Uysal-Biyikoglu and Gamal (2004), an uplink transmission scheduling problem is formulated and the “Flow Right” algorithm is proposed to solve the problem iteratively. The uplink scheduling problem is revisited in Miao and Cassandras (2006) where a number of structural properties of the optimal sample path is revealed. Mao and Cassandras further extend the work in Mao et al. (2007) to multiple stages, and the “Virtual Deadline Algorithm” (Mao and Cassandras 2009) is developed to provide a recursive solution that converges to the optimal control. The work in Mao and Cassandras (2009) is further extended to a multi-layer network where each layer also contains multiple nodes with multiple inputs and outputs (Mao 2014). The idea of “virtual deadline” is used again in Mao (2014) to develop a multilayer virtual deadline algorithm (MLVDA) that guarantees end-to-end real-time constraints.

In this paper, we consider the optimal scheduling problem for prioritized parallel queues in the general setting of Eq. (1). Scheduling for parallel queues has been studied in Tassiulas and Ephremides (1993), Sethuraman and Squillante (1999), Squillante et al. (2001), Xie and Lu (2015), and Xie et al. (2016). The major difference between our model and the existing ones is that in our formulation, *each* task has a real-time constraint associated with it.

Real-time scheduling has been widely studied in various contexts for both preemptive and nonpreemptive policies, and for various task arrival scenarios (Li et al. 2007; Aydin et al. 2004; Hong et al. 1999; Kim et al. 2002; Yao et al. 1995; Anderson et al. 2016). In this work, we consider a model that involves prioritized parallel queues with queue-dependent real-time constraints, and each queue is FCFS. Our model differs from the existing ones in the literature and has applications in computing, communications, and manufacturing. The contributions of this paper include: (i) We discover several structural properties of the optimal sample paths, (ii) We show that these structural properties can reduce the search space of the brute-force search; (iii) We propose a greedy algorithm which is shown near optimal numerically; and (iv) We design a receding horizon controller with good performance for on-line control. Some results of this paper are previously published in a conference paper (Miao 2010). What is new in this journal version is that we improve the proofs and also discuss on-line controller design.

The organization of this paper is as follows: in Section 2, we show the system model and formulate the optimization problem; Section 3 contains the structural properties of the optimal sample path; a greedy algorithm for off-line control is presented in Section 4; in Section 5, we study on-line controller design; finally, we conclude and discuss future work in Section 6.

## 2 System model and problem formulation

We consider DESs that contain a single server serving two prioritized queueing systems. See Fig. 1 for an illustration. Let  $a_{ij}$ ,  $v_{ij}$ , and  $x_{ij}$  be the arrival time, number of bits, and departure time of task  $j$  in queue  $i$  respectively ( $i = 1$  or  $2$ ). For applications which must maintain operational simplicity, we assume a FCFS and nonpreemptive queueing model for each queue. We also assume that the server can only serve one task at a time. Each task is served at a constant and yet controllable rate. Although our model is applicable to various scenarios in computing, manufacturing, and material handling, our assumptions are especially valid in computer communication applications due to the following three reasons: (i) In communication networks, it is often required to keep the sequence of the incoming packets and send them out in the same order, (ii) It is very costly or simply impossible to preempt and suspend a packet on the fly, and (iii) Communication tasks are usually transmitted one at a time and the transmission rate of each task often needs to be predetermined before the transmission starts.

We consider the queues are prioritized and the priority of each queue is reflected on task deadlines. Specifically, we assume each task  $j$  in queue  $i$  has a hard deadline  $a_{ij} + d_i$ , i.e., task  $j$  must depart the system  $d_i$  seconds after its arrival. The queue with higher priority has

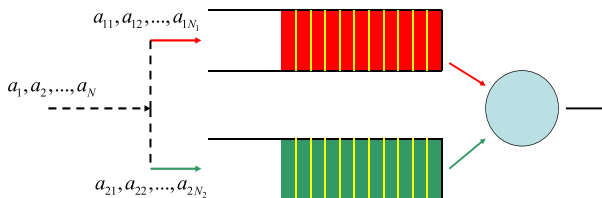


Fig. 1 System Model

a smaller  $d_i$ , which means tasks in a higher priority queue have tighter deadlines. We assume that  $d_1 \leq d_2$ , i.e., queue 1 has higher priority than queue 2. The prioritized queueing systems studied by this paper are widely used in communication networks. For instance, in Avaya’s Ethernet Routing Switch 8600, prioritized queues are used on each Egress (outgoing) port to differentiate critical network control traffic and high priority data traffic from low-priority data traffic.

Assume that we have totally  $N$  tasks in both queues. Let us order all the tasks by their arrival time. In order to ease our analysis, we introduce an imaginary data source (shown in dashed lines in Fig. 1). We can then imagine that the ordered sequence of  $N$  tasks from the imaginary data source are assigned to either queue 1 or queue 2, depending on their task priorities.

Let  $a_k, k = 1, \dots, N$ , be the arrival time of the  $k$ th task from the imaginary data source. Each  $a_k$  also corresponds to a task arrival time,  $a_{ij}$ , as illustrated in Fig. 2. Again, the deadline of each task  $a_{ij}$  is  $a_{ij} + d_i, i = 1$  or  $2$ . We use  $p_k \in \{0, 1\}$  to indicate the priority of task  $k$  ( $0$  means lower priority and  $1$  means higher priority). If all the tasks are served based on the order they arrive from the imaginary data source, then we can formulate the following optimization problem:

$$\mathbf{P1} : J_{1,N}^*(\mathbf{a}, \mathbf{D}, \mathbf{p}, \mathbf{v}, \theta) = \min_{\tau_1, \dots, \tau_N} \sum_{k=1}^N v_k \theta(\tau_k)$$

$$\text{s.t. } \tau_k \geq \tau_{\min}, \quad k = 1, \dots, N$$

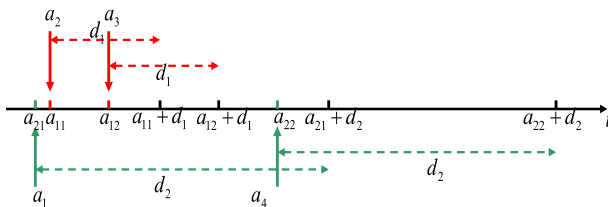
$$x_k = \max(x_{k-1}, a_k) + \tau_k v_k \leq D_k,$$

$$D_k = a_k + \mathbf{1}\{p_k = 1\}d_1 + \mathbf{1}\{p_k = 0\}d_2, \quad k = 1, \dots, N, \quad x_0 = 0$$

where  $\mathbf{a}$  is the arrival times,  $\mathbf{D}$  is the deadlines,  $\mathbf{p}$  is the task priorities,  $\mathbf{v}$  is the tasks’ sizes (for communication tasks, the size is the number of bits),  $\tau_k$ , defined as the amount of service time per bit, is the control over task  $k$ ,  $\mathbf{1}\{\cdot\}$  is the indicator function,  $x_k$  is the departure time of task  $k$ , and  $\theta(\tau_k)$  is the cost per bit associated with control  $\tau_k$ . Note that  $\tau_{\min}$  is the lower bound of the control, and the first constraint essentially is about the maximum rate at which the tasks can be served. When the above problem is infeasible, we let  $J_{1,N}^*(\mathbf{a}, \mathbf{D}, \mathbf{p}, \mathbf{v}, \theta)$  be  $\infty$ .

**Assumption 1**  $\theta(\tau)$  is nonnegative, strictly convex, monotonically decreasing, differentiable and  $\theta(\tau) \rightarrow -\infty$  as  $\tau \rightarrow 0$ .

Assumption 1 is justified in Gamal et al. (2002) for wireless applications, and channel coding schemes supporting this assumption can be found in Uysal-Biyikoglu et al. (2002).



**Fig. 2** Illustration for the arrival process

This assumption is also justified in Mao et al. (2007) for DVS applications. Note that in **P1**, we set the control of each task,  $\tau_k$ , to be static during the entire service process of task  $k$ . This is based on the fact that under Assumption 1, there is no benefit of dynamically adjusting the control  $\tau_k$  when serving task  $k$  (Miao and Cassandras 2005). In addition, there exists applications (e.g., wireless communications) in which the service rate of each task must be constant.

Problem **P1** above is essentially a special case of the problems studied in Mao et al. (2007) and Miao and Cassandras (2017). Therefore, CTDA developed in Mao et al. (2007) can be readily applied here to solve it. Because CTDA is extremely efficient, **P1** can be solved quickly without incurring high computational overhead. Note that Assumption 1 is the key assumption of CTDA, and the optimal solution to **P1** is independent of the exact forms of the cost function  $\theta(\tau)$  (Mao et al. 2007).

At each time instance, the server essentially needs to decide which queue to serve and what rate it should use. Our goal is to minimize the total cost of serving  $N$  tasks in the setting shown in Fig. 1 and also guarantee the real-time constraint for each task. The correct order that the server uses to serve these tasks is crucial. This is because once the order is known, the original problem can then be reduced to something similar to **P1** and solved by CTDA. Let  $\mathcal{P}$  be the set of all permutations of the original  $N$  tasks and  $q^z$  be the  $s$ -th permutation,  $z = 1, 2, \dots, 2^{N-1}$ . For the  $k$ -th task in the  $z$ -th permutation, let  $g_k^z$  be the task that it corresponds to in the original task arrival sequence. We formulate the following optimization problem:

$$\mathbf{P2} : \min_{q^z \in \mathcal{P}, z=1,2,\dots,2^{N-1}} J_{1,N}^*(\bar{\mathbf{a}}^z, \bar{\mathbf{D}}^z, \bar{\mathbf{p}}^z, \bar{\mathbf{v}}^z, \theta)$$

where  $\bar{\mathbf{a}}^z$ ,  $\bar{\mathbf{D}}^z$ ,  $\bar{\mathbf{p}}^z$ , and  $\bar{\mathbf{v}}^z$  are the arrival times, deadlines, priorities, and the sizes, respectively, of the reordered tasks; in particular,

$$\begin{aligned} \bar{a}_k^z &= \max\left(\bar{a}_{k-1}^z, a_{g_k^z}^z\right), \bar{a}_0^z = 0 \\ \bar{p}_k^z &= p_{g_k^z} \\ \bar{D}_k^z &= a_{g_k^z}^z + \mathbf{1}\{p_{g_k^z}^z = 1\}d_1 + \mathbf{1}\{p_{g_k^z}^z = 0\}d_2 \\ \bar{v}_k^z &= v_{g_k^z}, \quad k = 1, \dots, N \end{aligned}$$

Note that in the reordered sequence, a task’s deadline, priority, and the number of bits are unchanged. The only thing that may change is the arrival time: we manipulate the arrival time so that in the reordered sequence, the task arrival time is non-decreasing with the task’s sequence number. This is to maintain the FCFS nature of the DESs so that CTDA works on the reordered sequence; it does not have any effect on the optimal solution. In **P2** above, superscript  $z$  is used to denote the  $z$ -th permutation. For notation ease, we drop the superscript in subsequent analysis when it is clear what task sequence is being discussed.

Although the tasks in each queue are FCFS, the optimal order they leave the server may not be the same as the one they come from the imaginary data source. In other words, the optimal sequence may not be the original task sequence. Take the tasks in Fig. 2 as an example. Because  $a_{21}$  and  $a_{11}$  are very close, and task 1 has a much longer deadline than that of task 2, it may be more beneficial not to serve task 1 immediately when it arrives; instead, the optimal control may need to keep the server remain idling until task 2 arrives.

Then, the optimal order of serving these tasks would be 2, 1, 3, . . . , instead of the original order 1, 2, 3, . . . .

Our goal is now clear: find the optimal order of tasks such that the total cost is minimized. Specifically, we are interested in minimizing the total cost of serving  $N$  tasks that arrive at both queues while satisfying the real-time constraint associated with each individual task. We will first focus on the *off-line* optimal control problem where all the task information is known to us a priori. We use  $Q_1$  and  $Q_2$  to denote the sets of tasks that arrive at queue 1 and queue 2, respectively. Let  $N_1$  be the cardinality of  $Q_1$  and  $N_2$  be the cardinality of  $Q_2$ .

### 3 Structural properties of some optimal sample paths

Our first observation is that the optimal solution to **P2** is not unique. Consider a simple example that the system only has one low priority task and one high-priority task, and the two tasks have the same size. In addition,  $a_1 = a_{21} = 0$ ,  $d_2 = 15$ ,  $a_2 = a_{11} = 5$ , and  $d_1 = 5$ . Suppose that the problem is feasible, the following two orders are both optimal as the same constant control  $\tau$  is used on both tasks for a total duration of  $10s$  in each case: (i) Start serving the low-priority task at  $t = 0$  and complete serving the high-priority task at time  $t = 10$  and (ii) Start serving the high-priority task at  $t = 5$  and complete serving the low priority task at time  $t = 15$ .

Looking at **P2** above, the number of possible permutations of the given  $N$  tasks increases exponentially with  $N - 1$ . This means that the brute-force search will not scale well, especially when  $N$  is large. This raises two questions: i) is it possible that we may improve the brute-force approach by confining the search space and ii) are there any alternatives to the brute-force approach?

Assuming that **P2** is feasible, we will answer the above two questions by discovering the structural properties of the optimal sample path of **P2**.

We first introduce the concept of *busy period*.

**Definition 1** A *busy period* (BP) is a segment of an optimal sample path that contains a contiguous set of tasks  $\{s, \dots, n\}$  and satisfies the following three conditions:  $\bar{x}_{s-1}^* < \bar{a}_s$ ,  $\bar{x}_n^* < \bar{a}_{n+1}$ , and  $\bar{x}_k^* \geq \bar{a}_{k+1}$ , for every  $k = s, \dots, n - 1$ .

As shown in Mao et al. (2007) and Miao and Cassandras (2017), the concept of BP defined above can often decompose the original complex problem into a number of simpler problems, one for each BP. Due to the complex nature of **P2**, however, busy periods of **P2** are not unique and cannot be easily identified as in Mao et al. (2007) and Miao and Cassandras (2017).

**Definition 2** A *super busy period* (SBP) is a segment of an optimal sample path that contains one or more BPs.

Similar to BPs, SBPs are not unique and may vary from one optimal sample path to another. However, the following lemma shows that SBPs can be helpful.

**Lemma 1** For task  $s = 2, \dots, N$  in the original task arrival sequence, let

$$k_1 = \arg \max_{k \in \{1, \dots, s-1\} \cap Q_1} \{a_k\}, \quad k_2 = \arg \max_{k \in \{1, \dots, s-1\} \cap Q_2} \{a_k\}.$$

If

$$D_{k_1} = a_{k_1} + d_1 < a_s \text{ and } D_{k_2} = a_{k_2} + d_2 < a_s, \tag{2}$$

then task  $s$  and the tasks that arrive before it must belong to different SBPs on any optimal sample path.

*Proof*  $D_{k_1} = a_{k_1} + d_1 < a_s$  and  $D_{k_2} = a_{k_2} + d_2 < a_s$  mean that the deadlines of tasks  $k_1$  and  $k_2$  are before the arrival of task  $s$ . Therefore, tasks  $k_1$  and  $k_2$  must be completed before  $a_s$ . Because tasks  $k_1$  and  $k_2$  are the latest tasks that arrive earlier than task  $s$  in queue 1 and queue 2 respectively, all tasks arrive before task  $s$  must also be served before  $a_s$ .  $\square$

Lemma 1 shows that although we cannot tell exactly how the SBPs look like, we are able to figure out what tasks are included in some common SBPs among all optimal sample paths (by identifying all the tasks that satisfy (2)). In turn, solving **P2** boils down to finding out the optimal task sequence within each of these SBPs, which is a simpler problem since the number of tasks within each SBP is less than  $N$ . After the optimal solutions to all such SBPs are available, we combine them to obtain the final optimal solution to the original problem **P2**.

In what follows, we will introduce a few other important structural properties of the optimal sample paths.

**Lemma 2** *There exists an optimal sample path on which high priority task  $a$  is served earlier than all tasks that arrive later in the original task arrival sequence.*

*Proof* Suppose that on optimal sample path  $p$ , there exists a set of tasks that arrive later than  $a$ , but are served earlier than  $a$ . Let task  $b$  be the one served right before  $a$  in this set. We use  $a_{1,i}$  and  $a_{k,j}$ ,  $k \in \{1, 2\}$ , to denote the arrival times of tasks  $a$  and  $b$ , respectively. Note that  $k = 1$  and  $k = 2$  correspond to the cases that task  $b$ 's priority is high and low, respectively. By assumption, we have  $a_{1,i} < a_{k,j}$ . We will next show that there exists another optimal sample path  $p'$ , on which task  $a$  is served before task  $b$ .

Let us assume that task  $a$  is served between  $t_{a1}$  and  $t_{a2}$  and task  $b$  is served between  $t_{b1}$  and  $t_{b2}$  on optimal sample path  $p$ . We have

$$a_{1,i} \leq t_{a1} < t_{a2} \leq a_{1,i} + d_1 \tag{3}$$

and

$$a_{k,j} \leq t_{b1} < t_{b2} \leq a_{k,j} + d_k \tag{4}$$

Because task  $b$  is served first on this optimal sample path, we have

$$t_{b1} < t_{b2} \leq t_{a1} < t_{a2} \tag{5}$$

Combing (3), (4), and (5) above, we obtain

$$a_{1,i} < a_{k,j} \leq t_{b1} < t_{b2} \leq t_{a1} < t_{a2} \leq a_{1,i} + d_1 < a_{k,j} + d_k \tag{6}$$

Note that the last inequality in Eq. 6 holds because  $d_1 \leq d_k$ . Let  $n$ -tuple  $\mathbf{T}$  be an ordered list of  $n$ ,  $n \geq 0$ , tasks served between  $t_{b2}$  and  $t_{a1}$  on the optimal sample path  $p$ . On optimal sample path  $p$ , tasks are executed in the order of  $(b, \mathbf{T}, a)$  from time  $t_{b1}$  to  $t_{a2}$ . Consider another sample path  $p'$ , on which everything, including the serving rate (control) of each task, is identical to  $p$ , except that the order of task execution between  $t_{b1}$  and  $t_{a2}$  is:  $(\mathbf{T}_1, a, b, \mathbf{T}_2)$ , where tuple  $\mathbf{T}_1$  includes the set of tasks in  $\mathbf{T}$  whose arrival times are no later than  $a_{k,j}$  (the arrival time of task  $b$ ), tuple  $\mathbf{T}_2$  includes the set of remaining tasks in  $\mathbf{T}$ , and tasks in  $\mathbf{T}_1$  and

$\mathbf{T}_2$  have the same order as the one in  $\mathbf{T}$ . Next, we show that sample path  $p'$  is feasible, i.e., both causality and real-time constraints are satisfied for tasks  $(\mathbf{T}_1, a, b, \mathbf{T}_2)$  on sample path  $p'$ :

- (i) Causality. Using (6), task  $a$  arrives before  $t_{b1}$ , and by assumption, all tasks in tuple  $\mathbf{T}_1$  arrive no later than  $a_{k,j}$  and  $t_{b1}$ . Therefore, causality is met for these tasks. The start execution time of task  $b$  and all tasks in  $\mathbf{T}_2$  on sample path  $p'$  is later than that on sample path  $p$ ; therefore, causality is also satisfied for those tasks.
- (ii) Real-time constraints. The start execution time of task  $a$  and tasks in  $\mathbf{T}_1$  are earlier on sample path  $p'$  than that on sample path  $p$ . Since the controls are the same, their real-time constraints are met. The deadlines of task  $b$  and tasks in  $\mathbf{T}_2$  are later than  $a_{1,i} + d_1$ , and all of them are finished before  $t_{a2}$ , which is no later than  $a_{1,i} + d_1$  according to Eq. 6. Therefore, their real-time constraints are also met.

We now conclude that sample path  $p'$  is feasible. Since its controls are identical to those of optimal sample path  $p$ ,  $p'$  must be an optimal sample path. In case that there are other tasks on sample path  $p'$  that arrive later than  $a$ , but are served earlier than  $a$ , we repeat the above procedure over and over again, until we come up with an optimal sample path on which all tasks that arrive after task  $a$  are also served after it. □

Lemma 2 indicates that a high-priority task should be served before all tasks that arrive after it, regardless of their priorities and sizes. As we will see later, this property is very important and can be used to reduce the possible reorderings for the brute-force search.

**Lemma 3** *Suppose that the  $k$ -th task in the original task arrival sequence is a low-priority one. If there are no task arrivals in the high priority queue within time interval  $[a_k, a_k + d_2]$ , then there exists an optimal sample path on which (i) all low-priority tasks that arrive before task  $k$  in the original task arrival sequence are served before task  $k$  and (ii) all low-priority tasks that arrive after task  $k$  in the original task arrival sequence are served after task  $k$ .*

*Proof* Suppose that the arrival time of the  $k$ -th task in the original task arrival sequence is  $a_{2,j}$ . We first prove Part (i) of the lemma. Suppose that on optimal sample path  $p$ , there exists a set of low-priority tasks that arrive before  $k$ , but are served after  $k$ . Let task  $a$  be the one served right after task  $k$  in this set.

Let  $a_{2,i}$  be the arrival time of task  $a$ . By assumption, we have  $a_{2,i} < a_{2,j} = a_k$ . We will show that there exists another optimal sample path  $p'$ , on which task  $a$  is served before task  $k$ .

Let us assume that task  $a$  is served between  $t_{a1}$  and  $t_{a2}$  and task  $k$  is served between  $t_{k1}$  and  $t_{k2}$  on optimal sample path  $p$ . We have

$$a_{2,i} \leq t_{a1} < t_{a2} \leq a_{2,i} + d_2 \tag{7}$$

and

$$a_{2,j} \leq t_{k1} < t_{k2} \leq a_{2,j} + d_2 \tag{8}$$

Because task  $k$  is served first on this optimal sample path, we have

$$t_{k1} < t_{k2} \leq t_{a1} < t_{a2} \tag{9}$$

Combing (7), (8), and (9) above, we obtain

$$a_{2,i} < a_{2,j} = a_k \leq t_{k1} < t_{k2} \leq t_{a1} < t_{a2} \leq a_{2,i} + d_2 < a_{2,j} + d_2 \tag{10}$$



Let  $n$ -tuple  $\mathbf{T}$  be an ordered list of  $n$ ,  $n \geq 0$ , tasks served between  $t_{k2}$  and  $t_{a1}$  on the optimal sample path  $p$ . On optimal sample path  $p$ , tasks are executed in the order of  $(k, \mathbf{T}, a)$  from time  $t_{k1}$  to  $t_{a2}$ . Consider another sample path  $p'$ , on which everything, including the serving rate (control) of each task, is identical to  $p$ , except that the order of task execution between  $t_{k1}$  and  $t_{a2}$  is:  $(\mathbf{T}_1, a, k, \mathbf{T}_2)$ , where tuple  $\mathbf{T}_1$  includes the set of tasks in  $\mathbf{T}$  whose arrival times are no later than  $t_{k1}$ , tuple  $\mathbf{T}_2$  includes the set of remaining tasks in  $\mathbf{T}$ , and tasks in  $\mathbf{T}_1$  and  $\mathbf{T}_2$  have the same order as the one in  $\mathbf{T}$ . Next, we show that same path  $p'$  is feasible, i.e., both causality and real-time constraints are satisfied for tasks  $(\mathbf{T}_1, a, k, \mathbf{T}_2)$  on sample path  $p'$ :

- (i) Causality. Using (10), task  $a$  arrives before  $t_{k1}$ , and by assumption, all tasks in tuple  $\mathbf{T}_1$  arrive no later than  $t_{k1}$ . Therefore, causality is met for these tasks. The start execution time of task  $k$  and all tasks in  $\mathbf{T}_2$  on sample path  $p'$  is later than that on sample path  $p$ ; therefore, causality is also satisfied for those tasks.
- (ii) Real-time constraints. The start execution time of task  $a$  and tasks in  $\mathbf{T}_1$  are earlier on sample path  $p'$  than that on sample path  $p$ . Since the controls are the same, their real-time constraints are met. Because all tasks in  $\mathbf{T}_2$  arrive between  $[t_{k1}, t_{a2}]$ , which is within  $[a_{2,j}, a_{2,j} + d_2]$ , i.e.,  $[a_k, a_k + d_2]$ , according to Eq. (10), it follows by assumption that all these tasks are low-priority ones. It then yields that the deadlines of task  $k$  and tasks in  $\mathbf{T}_2$  are later than  $a_{2,i} + d_2$ . Because all of them are actually finished before  $t_{a2}$ , which is no later than  $a_{2,i} + d_2$  according to Eq. (10), their real-time constraints are also met.

We now conclude that sample path  $p'$  is feasible. Since its controls are identical to those of optimal sample path  $p$ ,  $p'$  must be an optimal sample path. In case that there are other tasks on sample path  $p'$  that arrive earlier than  $k$ , but are served after  $k$ , we repeat the above procedure over and over again, until we come up with an optimal sample path on which all tasks that arrive before task  $k$  are also served before it.

We now prove Part (ii) of the lemma. First, we assume that there exists an optimal sample path  $q$  on which all low-priority tasks that arrive before task  $k$  are served before  $k$ . Such an optimal sample path can be constructed by following the procedures above for proving Part (i) of the lemma. Suppose that on optimal sample path  $q$ , there exists a set of low-priority tasks that arrive after  $k$ , but are served before  $k$ . Let task  $b$  be the one served right before  $k$  in this set. Let  $a_{2,m}$  be the arrival time of task  $b$ . By assumption, we have  $a_k = a_{2,j} < a_{2,m}$ . We will show that there exists another optimal sample path  $q'$ , on which task  $b$  is served after task  $k$ .

Let us assume that task  $b$  is served between  $t_{b1}$  and  $t_{b2}$  and task  $k$  is served between  $t_{k1}$  and  $t_{k2}$  on optimal sample path  $q$ . We have

$$a_{2,m} \leq t_{b1} < t_{b2} \leq a_{2,m} + d_2 \tag{11}$$

and

$$a_{2,j} \leq t_{k1} < t_{k2} \leq a_{2,j} + d_2 \tag{12}$$

Because task  $b$  is served first on this optimal sample path, we have

$$t_{b1} < t_{b2} \leq t_{k1} < t_{k2} \tag{13}$$

Combing (11), (12), and (13) above, we obtain

$$a_k = a_{2,j} < a_{2,m} \leq t_{b1} < t_{b2} \leq t_{k1} < t_{k2} \leq a_{2,j} + d_2 < a_{2,m} + d_2 \tag{14}$$

Let  $n$ -tuple  $\mathbf{T}$  be an ordered list of  $n$ ,  $n \geq 0$ , tasks served between  $t_{b2}$  and  $t_{k1}$  on optimal sample path  $q$ . On optimal sample path  $q$ , tasks are executed in the order of  $(b, \mathbf{T}, k)$  from time  $t_{b1}$  to  $t_{k2}$ . Consider another sample path  $q'$ , on which everything, including the serving rate (control) of each task, is identical to  $q$ , except that the order of task execution between  $t_{b1}$  and  $t_{k2}$  is:  $(\mathbf{T}_1, k, b, \mathbf{T}_2)$ , where tuple  $\mathbf{T}_1$  includes the set of tasks in  $\mathbf{T}$  whose arrival times are no later than  $t_{b1}$ , tuple  $\mathbf{T}_2$  includes the set of remaining tasks in  $\mathbf{T}$ , and tasks in  $\mathbf{T}_1$  and  $\mathbf{T}_2$  have the same order as the one in  $\mathbf{T}$ . Next, we show that same path  $q'$  is feasible, i.e., both causality and real-time constraints are satisfied for tasks  $(\mathbf{T}_1, k, b, \mathbf{T}_2)$  on sample path  $q'$ :

- (i) Causality. Using (14), task  $k$  arrives before  $t_{b1}$ , and by assumption, all tasks in tuple  $\mathbf{T}_1$  arrive no later than  $t_{b1}$ . Therefore, causality is met for these tasks. The start execution time of task  $b$  and all tasks in  $\mathbf{T}_2$  on sample path  $q'$  is later than that on sample path  $q$ ; therefore, causality is also satisfied for those tasks.
- (ii) Real-time constraints. The start execution time of task  $k$  and tasks in  $\mathbf{T}_1$  are earlier on sample path  $q'$  than that on sample path  $q$ . Since the controls are the same, their real-time constraints are met. Because all tasks in  $\mathbf{T}_2$  arrive between  $[t_{b1}, t_{k2}]$ , which is within  $[a_{2,j}, a_{2,j} + d_2]$ , i.e.,  $[a_k, a_k + d_2]$ , according to Eq. (14), it follows by assumption that all these tasks are low-priority ones. It then yields that the deadlines of task  $b$  and tasks in  $\mathbf{T}_2$  are later than  $a_{2,j} + d_2$ . Because all of them are actually finished before  $t_{k2}$ , which is no later than  $a_{2,j} + d_2$  according to Eq. (14), their real-time constraints are also met.

We now conclude that sample path  $q'$  is feasible. Since its controls are identical to those of optimal sample path  $q$ ,  $q'$  must be an optimal sample path. We also need to show that on optimal sample path  $q'$ , all low-priority tasks that arrive before task  $b$  are still served before task  $k$ . Looking at two sample paths  $q$  and  $q'$ , the only tasks whose execution orders are changed are  $k$ ,  $b$ , and the ones in  $\mathbf{T}$ . According to Eq. (14), the tasks that arrive before  $a_k$  must also arrive before  $t_{b1}$ . This implies that all the tasks that arrive before task  $k$  (if exist) can only be in  $\mathbf{T}_1$ , which is served before task  $k$  on sample path  $q'$ . Therefore, sample path  $q'$  still ensures that all tasks arrive before task  $k$  are served before task  $k$ . In case that there are other tasks on sample path  $q'$  that arrive later than  $k$ , but are served before  $k$ , we repeat the above procedure for Part (ii) over and over again, until we come up with an optimal sample path on which all tasks that arrive after task  $k$  are also served after it.  $\square$

Lemma 3 shows that if there is no high-priority arrivals within a low-priority task  $k$ 's arrival time and deadline, then task  $k$  does not have to switch order with any other low-priority ones on the optimal sample path.

**Lemma 4** *Suppose that the  $k$ -th task in the original task arrival sequence is a low-priority one. If there are no task arrivals in the high-priority queue within time interval  $[a_k, a_k + d_2]$ , then there exists an optimal sample path on which this task is the  $k$ -th task.*

*Proof* Invoking Lemma 3, there exists an optimal sample path  $p$  on which all low-priority tasks that arrive before the  $k$ -th task are served before it and all low-priority ones that arrive after it are served after it. Suppose that this optimal sample path  $p$  contains a set of high-priority tasks that arrive before task  $k$ , but are served after task  $k$ . We invoke Lemma 2 to

find another optimal sample path on which all these high-priority tasks are served before task  $k$ . The process is very similar to the one stated in the proof for Lemma 2, where task  $b$  is essentially task  $k$  in this lemma. Therefore, we omit the details here. Nonetheless, we need to point out that the process does not affect the order of the low-priority tasks before and after task  $k$ ; in particular, the ones arrive before  $k$  are still served before it, and the ones arrive after  $k$  are still served after it. This can be seen by looking at the place where  $\mathbf{T}_1$  is introduced in the proof of Lemma 2: it contains all the tasks arriving before task  $b$  and is also served before  $b$ . Finally, because the arrival times of all high-priority tasks that arrive after the task  $k$  are beyond its the deadline, task  $k$  must be served before these tasks to guarantee feasibility. Combining all the results above, there exists an optimal sample path on which all tasks arrive before task  $k$  are served before it and all tasks arrive after task  $k$  are served after it, i.e., the  $k$ -th task in the original task arrival sequence is also the  $k$ -th task on this optimal sample path.  $\square$

Lemma 4 shows that under some conditions, a task’s order in an optimal sample path is exactly the same as the one in the original task arrival sequence. This gives hope that we may decompose each SBP into smaller chunks separated by these special tasks. The next lemma shows another result that can be used to further reduce the search space of possible task reorderings.

**Lemma 5** *Suppose that the  $k$ -th task in the original task arrival sequence is a low-priority one. If the  $l$ -th task in the original task sequence is a high-priority task, and it arrives within time interval  $[a_k + d_2 - d_1, a_k + d_2]$ , then there exists an optimal sample path on which task  $k$  is served before  $l$ .*

*Proof* Suppose that the arrival time of  $k$ -th task in the original task arrival sequence is  $a_{2,i}$ . Let  $a_{1,j}$  be the arrival time of task  $b$ , which is a high-priority one arriving within time interval  $[a_k + d_2 - d_1, a_k + d_2]$  By assumption, we have  $a_k = a_{2,i} \leq a_{1,j} = a_l$ . Suppose that task  $b$  is served before task  $k$  in an optimal sample path  $p$ . We will show that there exists another optimal sample path  $p'$ , on which task  $k$  is served before task  $b$ . Because  $a_{1,j} \geq a_k + d_2 - d_1$ , we have

$$a_{1,j} + d_1 \geq a_k + d_2 = a_{2,i} + d_2 \tag{15}$$

Let us assume that task  $k$  is served between  $t_{k1}$  and  $t_{k2}$  and task  $l$  is served between  $t_{l1}$  and  $t_{l2}$  on optimal sample path  $p$ . We have

$$a_{2,i} \leq t_{k1} < t_{k2} \leq a_{2,i} + d_2 \tag{16}$$

and

$$a_{1,j} \leq t_{l1} < t_{l2} \leq a_{1,j} + d_1 \tag{17}$$

Because task  $l$  is served first on this optimal sample path, we have

$$t_{l1} < t_{l2} \leq t_{k1} < t_{k2} \tag{18}$$

Combing (15), (16), (17), and (18) above, we obtain

$$a_k = a_{2,i} \leq a_{1,j} = a_l \leq t_{l1} < t_{l2} \leq t_{k1} < t_{k2} \leq a_{2,i} + d_2 \leq a_{1,j} + d_1 \tag{19}$$

Let  $n$ -tuple  $\mathbf{T}$  be an ordered list of  $n$ ,  $n \geq 0$ , tasks served between  $t_{l2}$  and  $t_{k1}$  on the optimal sample path  $p$ . On optimal sample path  $p$ , tasks are executed in the order of  $(l, \mathbf{T}, k)$  from time  $t_{l1}$  to  $t_{k2}$ . Consider another sample path  $p'$ , on which everything, including the serving rate (control) of each task, is identical to  $p$ , except that the order of task execution between  $t_{l2}$  and  $t_{k1}$  is:  $(\mathbf{T}_1, k, l, \mathbf{T}_2)$ , where tuple  $\mathbf{T}_1$  includes the set of tasks in  $\mathbf{T}$  whose arrival times are no later than  $t_{l1}$ , tuple  $\mathbf{T}_2$  includes the set of remaining tasks in  $\mathbf{T}$ , and tasks in  $\mathbf{T}_1$  and  $\mathbf{T}_2$  have the same order as the one in  $\mathbf{T}$ . Next, we show that sample path  $p'$  is feasible, i.e., both causality and real-time constraints are satisfied for tasks  $(\mathbf{T}_1, k, l, \mathbf{T}_2)$  on sample path  $p'$ :

- (i) Causality. Using (19), task  $k$  arrives before  $t_{l1}$ , and by assumption, all tasks in tuple  $\mathbf{T}_1$  arrive no later than  $t_{l1}$ . Therefore, causality is met for these tasks. The start execution time of task  $l$  and all tasks in  $\mathbf{T}_2$  on sample path  $p'$  is later than that on sample path  $p$ ; therefore, causality is also satisfied for those tasks.
- (ii) Real-time constraints. The start execution time of task  $k$  and tasks in  $\mathbf{T}_1$  are earlier on sample path  $p'$  than that on sample path  $p$ . Since the controls are the same, their real-time constraints are met. Because all tasks in  $\mathbf{T}_2$  arrive between  $[t_{l1}, t_{k2}]$ , which is within  $[a_{1,j}, a_{1,j} + d_1]$  according to Eq. (19), it follows that all these tasks' deadlines are not earlier than  $a_{1,j} + d_1$ , regardless of their priorities. Because all of them are actually finished before  $t_{k2}$ , which is not later than  $a_{1,j} + d_1$  according to Eq. (19), their real-time constraints are also met.

We now conclude that sample path  $p'$  is feasible. Since its controls are identical to those of optimal sample path  $p$ ,  $p'$  must be an optimal sample path.  $\square$

Lemmas 4 and 5 show that there exists an optimal sample path on which a low-priority task  $k$  does not have to be served after high-priority arrivals that occur after  $a_k + d_2 - d_1$ . It also implies that high-priority tasks, whose deadlines are after a low-priority task's deadline, do not have to be executed before the low-priority task. This, however, does not mean Earliest Deadline First (EDF) provides the optimal sequence to  $\mathbf{P2}$ : one can come up with cases that a low-priority task needs to be served before a high-priority task whose deadline is earlier.

With the help of the above lemmas, when we search for an optimal reordering of the original task arrival sequence, we only need to consider the order of a low priority task  $k$  and the high-priority arrivals between  $a_k$  and  $a_k + d_2 - d_1$ : there exists an optimal sample path on which task  $k$  only possibly needs to switch order with these high-priority ones.

**Proposition 1** *When both queues have equal priority, i.e.,  $d_1 = d_2$ , the original task arrival sequence is an optimal solution to  $\mathbf{P2}$ , i.e., FCFS guarantees optimality.*

*Proof* Setting  $d_1 = d_2$  in Lemma 5, we obtain that for any task  $l$  in queue 1 that arrives after task  $k$  in queue 2, there exists an optimal sample path in which  $k$  is served before  $l$ . Since the two queues have equal priority, we can achieve the same result for the other case: for any task  $l$  in queue 2 that arrives after task  $k$  in queue 1, there exists an optimal sample path in which  $k$  is served before  $l$ . Therefore, there exists an optimal sample path in which the optimal task order is simply the order the tasks arrive.  $\square$

**Table 1** Results of bursty arrivals

Optimal	Original arrival sequence	Greedy algorithm	% greedy is optimal
1	1.2753	1.0146	47%

The above proposition shows that when the tasks in each queue have equal priority, FCFS guarantees optimality. In this case, the size of the search space for an optimal task reordering is reduced to 1.

#### 4 A greedy algorithm for off-line control

The structural properties of the optimal sample path help us reduce the total number of possible reorderings that contain the optimal task sequence. In this section, we propose a greedy algorithm to overcome the above shortcoming of the brute-force approach. This greedy algorithm takes advantage of the structural properties of the optimal sample paths, and our goal is to find a balance between the system cost and the speed of the algorithm. Since it is a greedy algorithm, we try to minimize the cost at each immediate step. In particular, the algorithm determines the order of each low-priority task at a time, by only allowing it to switch order with high-priority tasks, which arrive between its arrival time and  $d_2 - d_1$  seconds later, and finding the order that incurs minimal cost; all the following low-priority tasks' orders are not finalized yet and will be decided at later steps. When the greedy algorithm is used, the  $k$ -th low-priority task's order is determined at the end of step  $k$ , and the incurred cost is no more than that of the previous step. Step by step, we determine the order of all tasks eventually after  $N_2$  steps, where  $N_2$  is the number of low-priority tasks.

**Lemma 6** *The greedy algorithm needs to compare no more than  $N_2(N_1 + 1)$  different task sequences.*

*Proof* Consider the case that all the high-priority tasks arrive after all the low-priority ones, and the optimal sequence is the original task arrival sequence. In this worst case, we need to compare  $N_1 + 1$  different task sequences at each step of the greedy algorithm. There are totally  $N_2$  steps. Therefore, the maximum number of task sequences the greedy algorithm need to compare is  $N_2(N_1 + 1)$ .  $\square$

Simulation has been done to compare the costs incurred by the optimal task sequence (found by the brute-force approach), the original task arrival sequence, and the task sequence returned by the greedy algorithm. In particular, we first normalize the costs and then average them over the number of simulation runs. The results are shown in the first three columns of Tables 1 and 2. The last column of each table is the percentage of simulation runs in which the greedy algorithm returns the optimal solution.

**Table 2** Results of uniform arrivals

Optimal	Original arrival sequence	Greedy algorithm	% greedy is optimal
1	1.0658	1.0065	50%

In Table 1, the results are obtained from 100 simulation runs of bursty arrivals. In each simulation run, there are five bursts, with 2-8 tasks per burst. Each burst has equal probability to be either high priority burst or low priority burst. The time interval between adjacent bursts is uniformly distributed between 10s and 20s, and the inter-arrival time between two tasks within a burst is uniformly distributed between 0s and 0.2s. We set  $d_1=5s$  for high-priority tasks and  $d_2=15s$  for low-priority tasks. Table 2's setting is different from that of Table 1: in each simulation run, 25 task arrivals are uniformly distributed between 0s and 200s, and each task has equal chance to be either high-priority task or low priority task.

As we can see that in both tables, the solution returned by the greedy algorithm is close to the optimal solution, and the greedy algorithm returns the optimal solution in nearly 50% of the total simulation runs.

## 5 Receding horizon on-line control

In this section, we turn our attention to the on-line control for the prioritized DESs described in earlier sections. Different from off-line control, where all task information is known to the controller a priori, on-line controllers do not have full future task information. This creates two problems in on-line control: (i) Feasibility is hard to be guaranteed and (ii) Optimization is hard to be carried out. We propose a Receding Horizon Controller (RHC) for the on-line control of the class of prioritized DESs discussed in this paper. Such a RHC has been used for single-queue DESs with real-time constraints (Miao 2007), and it has been shown that RHC is very effective in on-line control scenarios where the stochastic information of the task arrival process is unknown. The list of properties of RHC and its performance evaluation results can be found in Miao (2007). Some new results of using RHC for real-time single-queue DESs can be found in Miao (2016).

Let us first briefly explain the idea of RHC. In a nutshell, RHC works recursively and is applied at each decision point. A decision point in RHC could be either the departure time or the arrival time of a task. In this paper, we let the decision point be the departure time of each task on the RH sample path. There is one exception though: if a task ends a BP on the RH sample path, then the next decision point is the arrival time of the next task (since the RHC does not have to act until then). In particular, we assume that the RH controller knows task arrival information within a RH window of  $H$  seconds from each decision point and nothing beyond the window. This RH window is often referred to as the *planning horizon*. The RH controller solves a smaller scale optimization problem over the planning horizon at each decision point and applies the control to the *action horizon*. The action horizon usually is smaller than the planning horizon, and in this paper, we let it contain the next task only.

Whereas the task sequence on the RH sample path is the same as the original task arrival sequence in Miao (2007), things are quite different in this paper. Specifically, the  $i$ -th task on the RH sample path may not be the  $i$ -th task in Problem **P1**. This is due to the fact that some low-priority tasks may be executed after the high-priority ones which arrive after them. For the  $i$ -th task on the RH sample path, let us use  $\tilde{g}_i$  to denote its original index in **P1**.

We now introduce some notations and formally describe the proposed RHC. Let  $\tilde{x}_i$  denote the departure time of the  $i$ -th task *evaluated* by the RH controller on the RH sample path when the planning horizon contains this task. Once again, this task may not be the  $i$ -th one in **P1**. If  $\tilde{x}_i$  is the *actual* departure time of task  $t$  on the RH sample path, then it is also a decision point. When task  $t + 1$  starts a new BP (i.e.,  $a_{\tilde{g}_{t+1}} > \tilde{x}_t$ ), then the RH controller

does not need to act until  $a_{\tilde{g}_{t+1}}$  rather than  $\tilde{x}_t$ ; for notational simplicity, we will still use  $\tilde{x}_t$  to represent the decision point for task  $t + 1$ . Let  $h_t$  denote the last task included in the RH window that starts at the current decision point  $\tilde{x}_t$ , i.e.,

$$h_t = \arg \max_{r \geq t} \{a_{\tilde{g}_r} : a_{\tilde{g}_r} \leq \tilde{x}_t + H\}$$

Let  $\tilde{\tau}_i$  be the control associated with task  $i$  which is determined by the RH controller for all  $i = t + 1, \dots, h_t$ . The values of  $\tilde{x}_i$  and  $\tilde{\tau}_i$  are initially undefined, and are updated at each decision point  $\tilde{x}_t$  for all  $i = t + 1, \dots, h_t$ . Control is applied to task  $t + 1$  only. That control and the corresponding departure time are the ones shown in the final RH sample path. In other words, for any given task  $i$ ,  $\tilde{x}_i$  and  $\tilde{\tau}_i$  may vary at different decision points, since optimization is performed based on different available information. It is only when task  $i$  is the next one at some decision point that its control and departure time become final.

If  $h_t = N$ , then the optimization procedure will be finished. In what follows, we consider the more interesting case when  $h_t < N$ . In this case, our action horizon at decision point  $\tilde{x}_t$  only contains task  $t + 1$ , i.e., the RH controller only applies control to task  $t + 1$ .

We now define the optimization problem that the RHC solves at each decision point  $\tilde{x}_t$ . Let  $\mathcal{P}_t$  be the set of all permutations of the tasks available in the planning horizon and  $q^z$  be the  $s$ -th permutation,  $z = 1, 2, \dots, 2^{h_t-t-1}$ . For the  $i$ -th task in the  $z$ -th permutation, let  $\tilde{g}_i^z$  be the task that it corresponds to in the original task arrival sequence. We formulate the following optimization problem:

$$\tilde{Q}(t + 1, h_t) : \min_{q^z \in \mathcal{P}_t, z=1,2,\dots,2^{h_t-t-1}} J_{t+1,h_t}^*(\tilde{\mathbf{a}}^z, \tilde{\mathbf{D}}^z, \tilde{\mathbf{p}}^z, \tilde{\mathbf{v}}^z, \theta)$$

where  $\tilde{\mathbf{a}}^z, \tilde{\mathbf{D}}^z, \tilde{\mathbf{p}}^z$ , and  $\tilde{\mathbf{v}}^z$  are the arrival times, deadlines, priorities, and the sizes, respectively, of the reordered tasks within the planning horizon:

$$\tilde{a}_i^z = \max(\tilde{a}_{i-1}^z, a_{\tilde{g}_i^z}), \tilde{a}_t^z = \tilde{x}_t \tag{20}$$

$$\tilde{p}_i^z = p_{\tilde{g}_i^z} \tag{21}$$

$$\tilde{D}_i^z = \min(\tilde{x}_t + H, a_{\tilde{g}_i^z} + \mathbf{1}\{\tilde{p}_i^z = 1\}d_1 + \mathbf{1}\{\tilde{p}_i^z = 0\}d_2) \tag{22}$$

$$\tilde{v}_i^z = v_{\tilde{g}_i^z}, i = t + 1, \dots, h_t \tag{23}$$

Although  $\tilde{Q}(t + 1, h_t)$  is similar to **P2**, there are two major differences. First,  $\tilde{Q}(t + 1, h_t)$  is a smaller scale optimization problem on the planning horizon only, whereas **P2** is the off-line problem that involves all the tasks  $\{1, \dots, N\}$ . Second, the task deadlines in  $\tilde{Q}(t + 1, h_t)$  are capped by  $x_t + H$ , the boundary of the RH window. This is to ensure feasibility (when the off-line problem **P1** is feasible) in the worst case: task  $h_t + 1$  arrives at  $x_t + H$  and needs to be served using the maximum speed; as a result, all tasks prior to task  $h_t + 1$  must be completed by  $x_t + H$ . See Miao (2007) for details about worst-case estimation.

We point out that the major difference between the RHC in this paper and the one in Miao (2007) is that we allow the change of task execution sequence in this paper. Thus, the results obtained in Miao (2007) cannot be applied to the settings here directly. Note that  $\tilde{Q}(t + 1, h_t)$  may be infeasible due to worst-case estimation. In this case, the RHC uses the FCFS policy and applies the maximum speed control to the next task. We point out that

when this happens, it does not mean that RHC will definitely return infeasible solution: at future decision points, the optimization problem may become feasible again as the RH window rolling forward. We actually will show next that RHC guarantees feasibility, one of the properties of the proposed RHC for prioritized DESs.

**Lemma 7** *If the off-line problem **P1** is feasible, then the control returned by the RHC is also feasible.*

*Proof* We use induction to prove it.

- Step 1:* We consider the first decision point  $\tilde{x}_0$ . There are two cases:
  - Case 1.1:* Problem  $\tilde{Q}(1, h_0)$  is feasible. In this case, we have  $\tilde{x}_1 \leq \tilde{D}_1 \leq \bar{D}_1$ .
  - Case 1.2:* Problem  $\tilde{Q}(1, h_0)$  is infeasible. In this case, the RHC will apply the fastest control to task  $t + 1$ . Because the off-line problem **P1** is feasible, we also have  $\tilde{x}_1 \leq \tilde{D}_1 \leq \bar{D}_1$ .
- Step 2:* Suppose that  $\tilde{x}_{t+1} \leq \tilde{D}_{t+1} \leq \bar{D}_{t+1}$ , we show  $\tilde{x}_{t+2} \leq \tilde{D}_{t+2} \leq \bar{D}_{t+2}$ . Note that  $\tilde{x}_{t+1}$  and  $\tilde{x}_{t+2}$  are finalized at decision points  $\tilde{x}_t$  and  $\tilde{x}_{t+1}$ , respectively. We consider two cases at decision point  $\tilde{x}_{t+1}$ .
  - Case 2.1:* Problem  $\tilde{Q}(t + 2, h_{t+1})$  is feasible. In this case, we have  $\tilde{x}_{t+2} \leq \tilde{D}_{t+2} \leq \bar{D}_{t+2}$ .
  - Case 2.2:* Problem  $\tilde{Q}(t + 2, h_{t+1})$  is infeasible. We consider two subcases.
    - Case 2.2.1:* All  $\tilde{Q}$  problems are infeasible at previous decision points. This means that the RHC has to apply the maximum speed control for all tasks in  $\{1, \dots, t + 2\}$ . Since the off-line problem **P1** is feasible,  $\tilde{x}_{t+2} \leq \bar{D}_{t+2}$ .
    - Case 2.2.2:* Before  $\tilde{x}_{t+1}$ , there exists at least one decision point at which the  $\tilde{Q}$  problem is feasible. Let us assume that the last such decision point is  $\tilde{x}_i$ ,  $i \in \{0, \dots, t\}$ . We once again consider two subcases.
      - Case 2.2.2.1:*  $t + 2 \leq h_i$ . In this case, task  $t + 2$  is within the planning horizon at decision point  $\tilde{x}_i$ . Since  $\tilde{Q}(i + 1, h_i)$  is feasible and the maximum speed control has been applied to tasks  $\{i + 2, \dots, t + 2\}$ , we have  $\tilde{x}_{t+2} \leq \bar{D}_{t+2}$ .
      - Case 2.2.2.1:*  $t + 2 > h_i$ . In this case, task  $t + 2$  is beyond the planning horizon at decision point  $\tilde{x}_i$ . Since  $\tilde{Q}(i + 1, h_i)$  is feasible and the maximum speed control is applied to tasks  $\{i + 2, \dots, h_i\}$ , all tasks in  $\{i + 1, \dots, h_i\}$  are finished before  $\tilde{D}_{h_i}$ . Because the arrival time of task  $h_i + 1$  is greater than  $\tilde{D}_{h_i}$ , RHC applies maximum speed control to tasks  $\{h_i + 1, \dots, t + 2\}$ , and the off-line problem **P1** is feasible, it follows that  $\tilde{x}_{t+2} \leq \bar{D}_{t+2}$ .

□

We point out that finding the optimal reordering of tasks  $t + 1, \dots, h_t$  at each decision point  $x_t$  is non-trivial. The reason is that the tasks in the RH planning horizon may have more than two different deadlines: (i) The backlogged tasks may have deadlines shorter than the original ones and (ii) The deadlines of the tasks arrive in the planning horizon may be shortened due to the restriction of RH window size  $H$ . Essentially, the RH control problem at each decision point  $\tilde{x}_t$  can be considered as a smaller scale off-line problem with multiple priority levels. In on-line settings where the speed of the controller is crucial, our strategies in response to the added complexity are two-fold: (i) We establish a result next to show that some tasks’ execution order in the planning horizon can be easily determined and (ii) We modify the greedy algorithm introduced in the previous section to make it work for the RH control.



**Lemma 8** *Suppose that at decision point  $\tilde{x}_t$ , there exists a set of tasks  $\{t + 1, \dots, h_t\}$  in the planning horizon, ordered by their arrival times. For  $i \in \{t + 1, \dots, h_t\}$  and  $j \in \{t + 1, \dots, h_t\}$ , if  $i < j$  and  $\tilde{D}_j = \tilde{x}_t + H$ , then task  $i$  is executed before task  $j$  in the optimal reordering of  $\tilde{Q}(t + 1, h_t)$ .*

*Proof* We consider three cases:

- Case 1:  $\tilde{D}_i - \tilde{a}_i < \tilde{D}_j - \tilde{a}_j$ . In this case, task  $i$ 's priority is higher than that of task  $j$ . Since task  $i$  also arrives before task  $j$ , we invoke Lemma 2 and conclude that task  $i$  is executed before task  $j$  in the planning horizon.
- Case 2:  $\tilde{D}_i - \tilde{a}_i = \tilde{D}_j - \tilde{a}_j$ . This corresponds to the case that the priorities of tasks  $i$  and  $j$  are the same. We consider two subcases.
  - Case 2.1: They are both high-priority tasks. Since task  $i$  arrives earlier than task  $j$ , we invoke Lemma 2 and obtain that task  $i$  is executed before task  $j$  in the planning horizon.
  - Case 2.2: They are both low-priority tasks. Since task  $j$  is the last one in the planning horizon and does not have any high priority ones following it, we invoke Lemma 3 to get that task  $i$  is served before  $j$ .
- Case 3:  $\tilde{D}_i - \tilde{a}_i > \tilde{D}_j - \tilde{a}_j$ . This corresponds to the scenario that task  $i$ 's priority is lower than task  $j$ 's. We evaluate two subcases.
  - Case 3.1:  $\tilde{a}_j \geq \tilde{D}_j$ . Task  $i$  must be executed before task  $j$ .
  - Case 3.2:  $\tilde{a}_j < \tilde{D}_j$ . Because  $\tilde{D}_j \leq \tilde{D}_j = \tilde{x}_t + H$ , we have

$$\begin{aligned} \tilde{D}_i - \tilde{D}_j + \tilde{a}_j &\leq \tilde{a}_j \leq \tilde{D}_j, \text{ which is equivalent to} \\ \tilde{a}_i + \tilde{D}_i - \tilde{a}_i - \tilde{D}_j + \tilde{a}_j &\leq \tilde{a}_j \leq \tilde{a}_i + \tilde{D}_i - \tilde{a}_i. \end{aligned}$$

Let  $\tilde{d}_i = \tilde{D}_i - \tilde{a}_i$  and  $\tilde{d}_j = \tilde{D}_j - \tilde{a}_j$ , we can rewrite the above inequality into:

$$\tilde{a}_i + \tilde{d}_i - \tilde{d}_j \leq \tilde{a}_j \leq \tilde{a}_i + \tilde{d}_i$$

Invoking Lemma 5, it yields that task  $i$  is executed before task  $j$  in the planning horizon.  $\square$

Lemma 8 has the following corollaries.

**Corollary 1** *Suppose that at decision point  $\tilde{x}_t$ , there exists a set of tasks  $\{t + 1, \dots, h_t\}$  in the planning horizon. If a set of consecutive tasks  $\{i, \dots, h_t\}$ ,  $t + 1 \leq i \leq h_t$ , exists such that  $\forall$  task  $j \in \{i, \dots, h_t\}$ ,  $\tilde{D}_j = \tilde{x}_t + H$ , then task  $j$  is also the  $j$ -th task in the optimal reordering at this decision point.*

*Proof* Invoking Lemma 8, all other tasks that arrive earlier than task  $j$  should be executed before task  $j$ . Therefore, tasks  $\{i, \dots, h_t\}$  conserve their orders in the optimal reordering at this decision point.  $\square$

**Corollary 2** *If  $H \leq d_1$ , then the original task arrival sequence is optimal in RHC.*

*Proof* Suppose that task  $i$  arrives in the planning horizon, i.e.,  $\tilde{a}_i > \tilde{x}_t$ . Using the definition of  $\tilde{D}_i$ , we get

$$\tilde{D}_i = \tilde{x}_t + H$$

Invoking Lemma 8, none of the tasks that arrive before task  $i$  are executed after task  $i$ . Therefore, the original task arrival sequence is optimal at each decision point.  $\square$

The corollary above shows that when the RH windows size  $H$  is small (not greater than  $d_1$ ), the future information does not really help much, and the RHC should serve the tasks

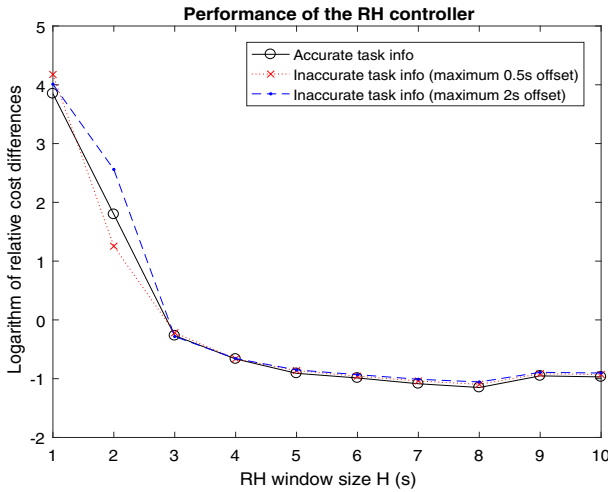


Fig. 3 The performance of the RH controller with uniform arrivals

based on their original order. When this happens, the optimization problem at each decision point becomes trivial: a constant speed that finishes all tasks within the planning horizon by  $x_t + H$  is optimal.

The modified greedy algorithm for RHC is presented in Algorithm 1. At each RH decision point, the controller solves a series of smaller scale optimization problems to select the next task to execute while the relative order of the rest of the tasks in the planning horizon is kept unchanged. In the worst-case, all  $N$  tasks are in the planning horizon, and the complexity of the RHC is  $O(N^2)$ . In what follows, we present some simulation results of the RH on-line controller. To quantify the deviation of the RH cost from the optimal off-line

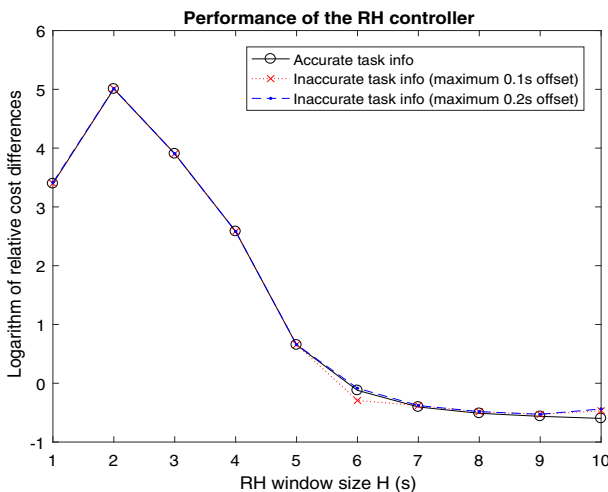


Fig. 4 The performance of the RH controller with bursty arrivals

cost, we define the relative cost difference as: (RH cost - optimal off-line cost) / optimal off-line cost. In our simulation, we consider both uniform (Fig. 3) and bursty (Fig. 4) arrivals where the settings are the same as those in Section 4. In both figures, we plot the logarithm of the relative cost difference against various RH windows sizes. There are three curves in each figure: one is obtained with accurate task information in the planning horizon, and the other two are the results of adding a random offset (up to a certain amount) to each task arrival in the planning horizon. RHC is known to have the ability to correct itself when the information used for optimization is deviated from the actual one; this is verified in our simulation results: all three curves in each figure match with each other closely, especially when the RH window size is large. It can be seen that in both scenarios, the RH cost is pretty close to the optimal cost when the RH window size  $H$  is close to  $10s$ . For uniform arrivals in Fig. 3, the RH controller always performs better when  $H$  is larger. For bursty arrivals in Fig. 4, the RH controller performs much better when  $H$  is greater than  $4s$ . However, when  $H$  is smaller than  $4s$ , a larger RH window size does not really help. This is caused by the fact that when  $H$  is between  $2s$  and  $4s$ , it is more likely to include a burst of tasks arriving near the boundary of the RH window; as a result, the worst-case estimation makes the RH controller use a high processing rate in order to finish all the tasks by  $x_t + H$ .

---

**Algorithm 1** Greedy algorithm for RHC at decision point  $\tilde{x}_t$

---

- 1: At decision point  $\tilde{x}_t$ , sort all unfinished tasks within the RH window by their arrival time
  - 2:  $\forall j \in \{t + 1, \dots, h_t\}$ , update  $\tilde{a}_j$ ,  $\tilde{D}_j$ ,  $\tilde{p}_j$ , and  $\tilde{v}_j$  using (20)–(23)
  - 3: **if**  $\exists j \in \{t + 1, \dots, h_t\}$ , s.t.,  $\forall l \in \{j, \dots, h_t\}$ ,  $D_l = \tilde{x}_t + H$  **then**
  - 4:      $i \leftarrow \arg \max_{j \geq t} \{\forall l \in \{j, \dots, h_t\}, D_l = \tilde{x}_t + H\}$
  - 5: **else**
  - 6:      $i \leftarrow h_t + 1$
  - 7: **end if**
  - 8:  $j \leftarrow t + 1$
  - 9: **for**  $j < i$  **do**
  - 10:      $l \leftarrow j$
  - 11:     **for**  $l < i$  **do**
  - 12:         Execute task  $l$  before  $j$  if task  $l$  has tighter deadline and it reduces costs
  - 13:          $l \leftarrow l + 1$
  - 14:     **end for**
  - 15: **end for**
  - 16: Apply the optimal RH control to task  $t + 1$  only
- 

## 6 Conclusions and future work

In this paper, we study a class of prioritized discrete event systems with real-time constraints. For off-line control, we discover structural properties of the optimal sample path of such DESs. Using the structural properties, we are able to reduce the search space of the optimal task execution sequence; a greedy algorithm is developed and shown to be near optimal numerically. For on-line control, we come up with a receding horizon controller and show that it is always feasible when the original task arrival sequence is feasible in the off-line setting. We also identify some properties of the RHC, which can be used to reduce the overhead of on-line computation. Finally, we modify the greedy algorithm for RHC and

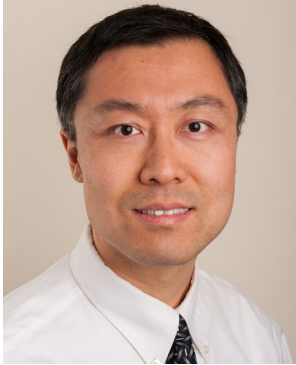
show via simulation that it is robust when the task information available in the planning horizon is deviated from the actual arrival time; it also has good performance when the RH window size is large.

Our future work includes studying the scenario that more than two prioritized queues exist. We believe that some structural properties of the optimal sample paths (e.g., Lemma 2) in Section 3 can be readily extended to the multi-queue case. Utilizing the structural properties, we think it is possible to come up with greedy algorithms that work well for the multi-queue case.

## References

- Anderson JH, Erickson JP, Devi UC, Casses BN (2016) Optimal semi-partitioned scheduling in soft real-time systems. *J Signal Process Syst* 84(1):3–23
- Aydin H, Melhem R, Mosse D, Mejia-Alvarez P (2004) Power-aware scheduling for periodic real-time tasks. *IEEE Trans Comput* 53:584–600
- Gamal AE, Nair C, Prabhakar B, Uysal-Biyikoglu E, Zahedi S (2002) Energy-efficient scheduling of packet transmissions over wireless networks. In: *Proceedings of IEEE INFOCOM*, vol 3, 23–27. New York City, pp 1773–1782
- Hong I, Kirovski D, Qu G, Potkonjak M, Srivastava M (1999) Power optimization of variable-voltage core-based systems. *IEEE Trans Comput-Aided Des Integr Circ Syst* 18(12):1702–1714
- Kim W, Shin D, Yun H, Kim J, Min SL (2002) Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In: *Real-Time and embedded technology and applications symposium*, pp 219–228
- Li W, Kavi K, Akl R (2007) A non-preemptive scheduling algorithm for soft real-time systems. *Comput Electr Eng* 33:12–29
- Mao J (2014) Optimal control of multilayer discrete event systems with real-time constraint guarantees. *IEEE Trans Syst Man Cybern Syst* 44(10):1425–1434
- Mao J, Cassandras CG (2009) Optimal control of multi-stage discrete event systems with real-time constraints. *IEEE Trans Autom Control* 54:108–123
- Mao J, Cassandras CG (2010) Optimal admission control of discrete event systems with real-time constraints. *Discret Event Dyn Syst* 20(1):37–62
- Mao J, Cassandras CG, Zhao Q (2007) Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints. *IEEE Trans Mob Comput* 6(6):678–688
- Miao L (2007) Receding horizon control for a class of discrete-event systems with real-time constraints. *IEEE Trans Autom Control* 52:825–839
- Miao L (2010) Structural properties of optimal scheduling in prioritized discrete event systems with real-time constraints. In: *2010 49th IEEE Conference on decision and control (CDC)*. IEEE, pp 6747–6752
- Miao L (2016) Receding horizon control with two planning horizons for a class of discrete event systems with real-time constraints. In: *2016 IEEE 55th Conference on decision and control (CDC)*. IEEE, pp 1939–1944
- Miao L, Cassandras CG (2005) Optimality of static control policies in some discrete event systems. *IEEE Trans Autom Control* 50:1427–1431
- Miao L, Cassandras CG (2006) Structural properties of optimal uplink transmission scheduling in energy-efficient wireless networks with real-time constraints. In: *Proceedings of the 45th IEEE conference on decision and control and european control conference*. San Diego, pp 2997–3002
- Miao L, Cassandras CG (2017) Optimal energy-efficient downlink transmission scheduling for real-time wireless networks. *IEEE Trans Control Net Syst* 4(4):692–706
- Sethuraman J, Squillante MS (1999) Optimal stochastic scheduling in multiclass parallel queues. *ACM SIGMETRICS Perform Eval Rev* 27:93–102
- Squillante MS, Xia CH, Yao DD, Zhang L (2001) Threshold-based priority policies for parallel-server systems with affinity scheduling. In: *Proceedings of the American control conference*, vol 4, Arlington, pp 2992–2999
- Tassioulas L, Ephremides A (1993) Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Trans Inf Theory* 39:466–478
- Uysal-Biyikoglu E, Gamal AE (2004) On adaptive transmission for energy efficiency in wireless data networks. *IEEE Trans Inf Theory* 50:3081–3094

- Uysal-Biyikoglu E, Prabhakar B, Gamal AE (2002) Energy-efficient packet transmission over a wireless link. *IEEE/ACM Trans Network* 10:487–499
- Xie Q, Lu Y (2015) Priority algorithm for near-data scheduling: throughput and heavy-traffic optimality. In: 2015 IEEE Conference on computer communications (INFOCOM). IEEE, pp 963–972
- Xie Q, Yekkehkhany A, Lu Y (2016) Scheduling with multi-level data locality: throughput and heavy-traffic optimality. In: IEEE INFOCOM 2016-The 35th Annual IEEE international conference on computer communications. IEEE, pp 1–9
- Yao F, Demers A, Shenker S (1995) A scheduling model for reduced cpu energy. In: Proceedings of the 36th annual symposium on foundations of computer Science, pp 374–382



**Lei Miao** received the B.S. and M.S. degrees from Northeastern University, Shenyang, Liaoning, China, and the Ph.D. degree from Boston University, Boston, MA, in 1998, 2001, and 2006, respectively. From 2006 to 2009, he was with Nortel Networks in Billerica, MA. From 2009 to 2011, he was visiting professor at the University of Cincinnati. From 2011 to 2014, he was with NuVo Technologies/Legrand North America in Hebron, KY. From 2014 to 2015, he was a faculty member at State University of New York Farmingdale. Currently, he is Assistant Professor of Mechatronics Engineering at Middle Tennessee State University. His research interests include control and optimization for Discrete Event Systems and hybrid systems, with applications to communication networks, wireless networks, and cyber-physical systems.