Distance-Constrained Scheduling and Its Applications to Real-Time Systems

Ching-Chih Han, *Member, IEEE*, Kwei-Jay Lin, *Member, IEEE Computer Society*, and Chao-Ju Hou, *Member, IEEE*

Abstract—In hard real-time systems, each task must not only be functionally correct but also meet its timing constraints. A common approach to characterizing hard real-time tasks with repetitive requests is the *periodic task* model [1]. In the periodic task model, every task needs to be executed once during each of its periods. The execution of a task in one period is independent of the execution of the same task in another period. Hence, the executions of the same task in two consecutive periods may be right next to each other, or at the far ends of the two periods. While the periodic task model can serve as a simple paradigm for scheduling tasks with repetitive requests, it may not be suitable for all real-time applications. For example, in some real-time systems, the temporal distance between the finishing times of any two consecutive executions of the same task must be less than or equal to a given value. In other words, each execution of a task has a deadline relative to the finishing time of the previous execution of the same task. Scheduling algorithms designed for the periodic task model may not provide efficient solutions for tasks with temporal distance constraints.

In this paper, we propose the (preemptive) distance-constrained task system model which can serve as a more intuitive and adequate scheduling model for "repetitive" task executions. We design an efficient scheduling scheme for the model, and derive a schedulability condition for the scheduling scheme. The schedulability condition is a measure for providing the fundamental predictability requirement in hard real-time applications. To show the usefulness of the distance-constrained task model and its scheduling scheme, we also discuss how to apply the scheduling scheme to real-time sporadic task scheduling and to real-time communications.

Index Terms—Distance-constrained task systems, (temporal) distance constraints, periodic task systems, pinwheel problem, real-time scheduling/communication.

1 Introduction

In hard real-time systems, each task must not only be functionally correct but also meet its timing constraints. For example, each task can only be invoked after its *ready time*, and must be completed before its *deadline*. A common approach to scheduling hard real-time tasks with repetitive requests is the *periodic task* model [1], in which each task T_i has a period P_i and an execution time e_i. T_i must be executed once in each of its periods. Execution of the task in any one period is scheduled independently of executions of the same task in other periods. That is, each execution, called a *job request* (or simply, a job), of a task has a fixed ready time and a fixed deadline, which are the beginning and the end of its period, respectively. Every job must start its execution after its ready time and completes before its deadline.

The periodic task model, however, does not suffice to characterize the timing requirements of all real-time systems. In some real-time systems, tasks must satisfy some relative timing relationships. For example, some real-time tasks must be executed in a (temporal) *distance-constrained* manner [2], [3], rather than just periodically. The temporal distance between any two consecutive executions of a task should not be longer than a certain amount of time. A real-life example that requires relative timing constraints between two consecutive jobs is the oil change schedule for a car. Suppose a car is to have its oil changed every six months, i.e., an oil change must be done within six months after the previous oil change. If the last oil change was done in January, the next oil change should be done by July. However, if we use the periodic task model to schedule oil changes with a (fixed) period of six months (starting from January), it is acceptable to have one oil change done in January and the next in December, since there is one oil change in each of the six month periods from January to June and from July to December. Note that the actual temporal distance between the two oil changes is 11 months, rather than six months. In other words, in the schedule for the oil change task obtained by an algorithm designed for the periodic task model, there may be a temporal distance between two consecutive executions longer than the task period.

Many real-time applications have distance constraint requirements, e.g., multimedia data transmissions [4], delay and jitter control in ATM (Asynchronous Transfer Mode) networks [5], [6], chemical process control, and air traffic control [7], just to name a few. For example, the *continuous*

- *C.-C. Han is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, MI 48109-2122. E-mail: cchan@eecs.umich.edu.*
- *K.-J. Lin is with the Department of Electrical and Computer Engineering, University of California at Irvine, Irvine, CA 92717. E-mail: klin@ece.uci.edu.*
- *C.-J. Hou is with the Department of Computer Engineering, the University of Wisconsin, Madison, WI 53706-1691. E-mail: jhou@ece.wisc.edu.*

Manuscript received Jan. 12, 1995; revised June 28, 1995.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96112.

0018-9340/96$05.00 ©1996 IEEE

or *isochronous* media data, such as CD audio or video, must be displayed/replayed under *relative* timing constraints: once sample k is played, sample $k + 1$ must be played no later than a fixed interval (e.g., 125 μs). Note that tasks in these applications can be viewed as preemptive ones: the transmission of media data, such as an MPEG [8] video frame, can be preempted, but the entire frame must be transmitted before a certain time (e.g., 1/30 sec.) after the previous frame has been transmitted. In an ATM network, each message is divided into basic transmission entities, called cells, and the transmission of a message can be preempted after the transmission of a cell. The end-to-end delays and the end-to-end delay jitters (i.e., the variances of the delays) of messages in a message stream must be less than or equal to some user-specified values. In other words, jitter control regulates the message arrival spacings at the destination node. In a chemical process control system, one ingredient must be added into a container within a certain time after another has been put in. When one schedules a moving cart to ship the ingredients, it is important to have the cart come in some regular interval so that all necessary ingredients are added in at the right time. In monitoring aircraft, one also needs to monitor aircraft in a regular interval rather than in some random pattern as in executing periodic tasks.

These types of timing constraints, which we call *temporal distance constraints* [2], [3], require an acceptable temporal distance between the times that two consecutive job requests of a task are executed. In other words, a job request of a task has a relative deadline depending on when its predecessor was actually executed. In fact, when a system requirement of having a certain task performed once in every P seconds, it is usually referred to having the maximum temporal distance between two consecutive task executions be less than or equal to P. The schedule obtained by using the periodic task model with a period P may not necessarily satisfy the above criterion. We, thus, propose a real-time task model, termed as the *Distance-Constrained Task System* (DCTS), to remedy the deficiency of the periodic task model. We also devise an efficient scheduling algorithm for DCTS with *preemptive* tasks, and derive a schedulability condition for the algorithm. The case with nonpreemptive tasks has been studied in [2], [3].

In this paper, we first define the distance-constrained task system model which can serve as a more intuitive and adequate scheduling model for "repetitive" task executions. Using techniques developed for the *pinwheel problem* [9], [10], we then devise a DC scheduling scheme which efficiently schedules distance-constrained task sets. We also derive a schedulability condition (density threshold) for the scheduling scheme. The density threshold serves as a measure for providing the fundamental predictability requirement in hard real-time applications, i.e., the schedulability of a task set can be checked by comparing the total density[1] of the task set with the density threshold. The derived density threshold turns out to be exactly the same as the utilization bound for scheduling periodic tasks using the well-known *rate-monotonic* algorithm [1]. Finally, to show the usefulness of the proposed DC scheduling scheme, we discuss how to apply it to a real-time scheduling and a real-time communication problems.

The rest of the paper is organized as follows. In Section 2, we formally define the DC scheduling problem, and briefly summarize the related work, namely, the periodic task system, the rate-monotonic scheduling algorithm, and the pinwheel problem. In Section 3, we discuss how to use the algorithms designed for periodic task systems and for the pinwheel problem to schedule DC task sets. In Section 4, we present the main results. In particular, we propose a fixed-priority DC scheduling algorithm for DC task sets and derive its schedulability condition. In Section 5, we discuss how to apply the proposed scheme to the sporadic task scheduling problem and to a real-time communication problem. The paper concludes with Section 6.

2 DC Scheduling Problem and Related Work

In this section, we first formally define the distance-constrained task system (DCTS) model and the DC scheduling problem. Then, we briefly describe two closely related scheduling problems: the periodic task scheduling problem [1], [11] and the pinwheel problem [9], [10]. The scheduling algorithms originally designed for periodic task systems and for the pinwheel problem will be integrated to schedule distance-constrained task systems in Section 4.

2.1 DC Scheduling Problem

Formally, a distance-constrained task system (DCTS) consists of a set of n distance-constrained (DC) tasks $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$. Each task must be executed (virtually) an infinite number of times. Each execution of a task is called a *job*, and the jobs of T_i are denoted as $J_{i1}, J_{i2}, J_{i3}, \cdots$. Task T_i has an execution time e_i and a (temporal) distance constraint c_i. The *distance* between two (consecutive) jobs of a task is defined to be the difference of the finishing times of these two jobs. That is, if f_{ij} denotes the finishing time of job J_{ij}, for $1 \leq i \leq n$ and $j \geq 1$, the distance between J_{ij} and $J_{i,j+1}$ is defined to be $f_{i,j+1} - f_{ij}$. The distance constraint c_i imposes that $f_{i1} \leq c_i$ and $f_{i,j+1} - f_{ij} \leq c_i$, for all $j \geq 1$. In addition, we assume that there is a precedence constraint between two consecutive jobs, i.e., job $J_{i,j+1}$ can be started only after job J_{ij} has been finished, and that each job is ready to be executed as soon as its predecessor job is finished. Hence, job J_{i1} has a *ready time* $r_{i1} = 0$ and a *deadline* $d_{i1} = c_i$, and job J_{ij}, for $j > 1$, has a ready time $r_{ij} = f_{i,j-1}$ and a deadline $d_{ij} = f_{i,j-1} + c_i$. Moreover, we assume that jobs are *preemptable*, i.e., the execution of a job can be suspended at any time and resumed at a later time.

A scheduling algorithm which produces a schedule for a set of tasks in which each job J_{ij} starts its execution after its ready time r_{ij} and completes before its deadline d_{ij} is said to *feasibly schedule* the task set, and the schedule produced is said to be *feasible*.

A scheduling algorithm is said to be *optimal* for a particular scheduling scheme, if for any task set that can be feasibly scheduled by any other algorithm of the same scheme, it can also be feasibly scheduled by the algorithm.

[1] To be defined later.

2.2 Periodic Task Scheduling Problem

The problem of scheduling real-time periodic task systems has been extensively studied [1], [11], [12], [13], [14], [15]. Similar to the definition of DCTS, a real-time periodic task system consists of a set of n periodic tasks $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$. Each task T_i must be executed once every P_i time units, where P_i is called the period of T_i. Each execution of a periodic task is called a *job*, and each task T_i has an execution time e_i. The jth job J_{ij} of task T_i is ready for execution at time $(j-1) \cdot P_i$, and must be completed by the starting time $j \cdot P_i$ of the next job period of the same task, i.e., the *ready* (or *request*) *time* and the *deadline* of J_{ij} are $r_{ij} = (j-1) \cdot P_i$ and $d_{ij} = j \cdot P_i$, respectively. Note that the only difference between a periodic task and a DC task is that each job J_{ij} of a periodic task T_i has a fixed ready time $(j-1) \cdot P_i$ and a fixed deadline $j \cdot P_i$, while each job J_{ij} of a DC task T_i has a ready time $f_{i,j-1}$ and a deadline $f_{i,j-1} + c_i$, where $f_{i,j-1}$ is the finishing time of the predecessor job, $J_{i,j-1}$, of J_{ij}, i.e., the ready time and the deadline of J_{ij} are relative to the finishing time of $J_{i,j-1}$.

A commonly-used on-line scheduling scheme for real-time periodic task systems is the *priority-driven* preemptive scheduling scheme. In priority-driven scheduling, each task is given a (fixed or dynamic) priority. At run-time, the system always chooses among the *active* jobs the one with the highest priority to execute next, where an *active* job is one whose execution has been requested but not yet finished. If each task is given a fixed priority for all of its executions (jobs), the scheduling scheme is said to be *fixed* or *static* priority-driven. On the other hand, if the priority of a task changes from one execution to another (i.e., each job of the task has its own distinct priority), the scheduling scheme is said to be *dynamic* priority-driven.

Liu and Layland [1] showed that the *rate-monotonic* (RM) algorithm is optimal for fixed priority-driven scheduling schemes, and the *deadline-driven* algorithm, or termed elsewhere the *earliest-deadline-first* (EDF) algorithm, is optimal for dynamic priority-driven scheduling schemes. The RM algorithm assigns priorities to tasks according to their periods. Tasks with shorter periods get higher priorities. Since the periods of tasks are fixed, their priorities are also fixed. The EDF algorithm assigns priorities to tasks according to the deadlines of the current job requests. The earlier a job's deadline is, the higher its priority is. Therefore, the priority of a task changes with time. We may also say that the EDF algorithm assigns a fixed priority to each job, instead of to each task. Jobs with earlier deadlines get higher priorities.

Liu and Layland further defined e_i/P_i as the *processor utilization* of task T_i, and showed that

1) a task set can be feasibly scheduled by the EDF algorithm if and only if the *total utilization factor*

$$U_{\mathrm{T}} = \sum_{i=1}^{n} e_i/P_i$$

is less than or equal to one, and

2) the least upper bound for a task set to be feasibly scheduled by the RM algorithm is $K(n) = n(2^{1/n} - 1)$, i.e., if

$$U_{\mathrm{T}} = \sum_{i=1}^{n} e_i/P_i \le K(n) = n\left(2^{1/n} - 1\right)$$

then the task set is guaranteed to be schedulable by the RM algorithm. Note, however, that if the utilization factor U_{T} is larger than $K(n)$, it is not known whether or not the task set T can be feasibly scheduled by the RM algorithm.

2.3 Pinwheel Problem

Another scheduling problem closely related to our DC scheduling problem is the *pinwheel problem* defined below [9], [10].

PROBLEM 1. **(The Pinwheel Problem)** Given a multiset of n positive integers $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$, find an infinite sequence (schedule) over the symbols $\{1, 2, \ldots, n\}$ such that there is at least one symbol "i" within any subsequence of a_i consecutive symbols (slots).

For example, given a multiset $\mathbf{A} = \{2, 4, 5\}$, one solution sequence is $(1, 2, 1, 3, 1, 2, 1, 3, \cdots)$ where the subsequence $(1, 2, 1, 3)$ repeats forever. In this solution sequence, we can find one "1" in every $a_1 = 2$ consecutive symbols, one "2" in every $a_2 = 4$ consecutive symbols, and (at least) one "3" in every $a_3 = 5$ consecutive symbols. Also note that any two consecutive occurrences of symbol "i" have a temporal distance less than or equal to a_i, for $i = 1, 2$, and 3.

The pinwheel problem is motivated by the satellite communication with a single ground station. The ground station receives messages transmitted from several satellites. During each time slot, the ground station can only receive messages from at most one satellite. Each satellite i is associated with a transmission repetition interval a_i. Satellite i transmits the same message for a_i consecutive time slots, and then transmits another message for the next a_i consecutive time slots. This pattern repeats (virtually) forever. In order not to miss any message transmitted from the satellites, the ground station must schedule to receive the messages from satellite i at least once in any a_i consecutive time slots.

The pinwheel problem can be viewed as a special case of the discrete version of the distance-constrained scheduling problem since the pinwheel requirement that "there is at least one symbol 'i' within any subsequence of a_i consecutive symbols" is equivalent to the distance constraint that "the temporal distance between two consecutive occurrences of symbol 'i' must be less than or equal to a_i." A DC scheduling problem instance $\{T_i = (e_i, c_i) \mid 1 \le i \le n\}$ with all execution times $e_i = 1$ and all distance constraints c_i being integers is exactly the same as the pinwheel problem instance $\{a_i = c_i \mid 1 \le i \le n\}$.

In the DC scheduling problem, e_i/c_i is only the lower bound of the processor utilization of task T_i, and hence, we will follow the terminology used in the pinwheel problem and call $\rho(T_i) = e_i/c_i$ the *density* of task T_i. The *total density* of task set $\mathbf{T}$ is thus defined to be

$$\rho(\mathbf{T}) = \sum_{i=1}^{n} \rho(T_i) = \sum_{i=1}^{n} \frac{e_i}{c_i}.$$

Chan and Chin [10], [16] have devised several algorithms for scheduling pinwheel instances, including Schedulers **Sa**, **Sx**, **Sbc**, **Sby**, and **Sxy**. Their density thresholds (schedulability conditions) for guaranteeing a feasible schedule for a pinwheel problem instance have also been derived, to be 1/2, 13/20, 2/3, 0.6964, and 0.7, for **Sa**, **Sx**, **Sbc**, **Sby**, and **Sxy**, respectively. In Sections 3 and 4, we show how to use or modify these scheduling algorithms to schedule DC task sets.

3 Applying Known Scheduling Algorithms to DCTS

In this section, we show why the previously-known scheduling algorithms for the periodic task systems and for the pinwheel problem, though they can be straightforward applied to scheduling DC task sets, are not good solutions to the DC scheduling problem.

If a job is allowed to be arbitrarily preempted, a distance-constrained task system can be easily scheduled (as long as $\sum_{i=1}^{n} \frac{e_i}{c_i} \le 1$) using the following method: We first divide every task T_i into two parts T_i' and T_i''. The first part T_i' has an execution time $e_i - \varepsilon$ and the second part T_i'' has an execution time ε. We can choose $\varepsilon > 0$ to be arbitrarily small, and schedule J_{ij}'' (the jth job of T_i'') at time $j \cdot c_i - \varepsilon$, for all i and j (see Fig. 1). We then treat the rest of the tasks T_i's as a periodic task system. Each task T_i' has an execution time $e_i - \varepsilon$ and a period $P_i = c_i$. In this way, we can feasibly schedule all these tasks using the EDF algorithm as long as $\sum_{i=1}^{n} \frac{e_i - \varepsilon}{c_i} \le 1 - \xi$, where ξ is a small number depending on ε. Moreover, the distance constraints are all satisfied. In other words, if arbitrary preemption is allowed, a distance-constrained task set can be trivially scheduled by the EDF algorithm. However, in most, if not all, real-time applications, this kind of "arbitrary" preemption is meaningless, or even infeasible. Therefore, we will henceforth exclude this kind of arbitrary preemption and consider only "reasonable" preemptive scheduling schemes.

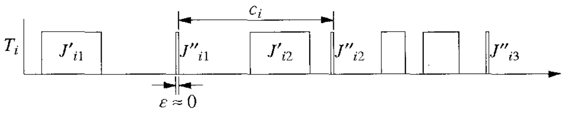


Fig. 1. Scheduling distance-constrained tasks that allow arbitrary preemption.

The other scheduling algorithms designed for periodic task systems can also be used to schedule distance-constrained task systems. Recall that in the periodic task system, a task must be executed once in each of its periods. Two consecutive job executions of a task hence have an average distance equal to the period of the task. However, as discussed earlier, there is no guarantee on when the two consecutive jobs of a periodic task are actually executed. Consider Fig. 2 for example: Given a periodic task T_i with a period P_i and an execution time e_i, the actual executions (finishing times) of two consecutive jobs J_{ij} and $J_{i,j+1}$ of the task can be as close as e_i, and as far as $2P_i - e_i$. This is because the latest finishing time of a job is the end of its period (i.e., the starting time of the next job period of the same task) and the earliest finishing time of its successor job is the starting time of the next job period plus the execution time of the task. Similarly, the earliest finishing time of a job is the starting time of its period plus the execution time of the task and the latest finishing time of its successor job is the end of the next job period. If we treat a DC task set as a periodic task set and use the EDF algorithm to schedule the set, the period P_i of task T_i must be set to a value that satisfies the following condition:

$$P_i \le (c_i + e_i)/2$$

in order to guarantee that a task T_i always meets its distance constraint c_i. That is, if we set $(c_i + e_i)/2$ as the period P_i of task T_i, for all i, in the distance-constrained task system and treat the system as a periodic task system, then we can schedule the system using the EDF algorithm to satisfy the distance constraints as long as $\sum_{i=1}^{n} e_i/P_i = \sum_{i=1}^{n} \frac{2e_i}{c_i+e_i} \le 1$. This result immediately gives the 0.5 density threshold derived in [9], [10], i.e., all instances of the pinwheel problem with a total density no more than 0.5 can always be feasibly scheduled. The major drawback of this approach is that in the resulting schedule, the tasks will be executed too often, leading to a waste of system resources.

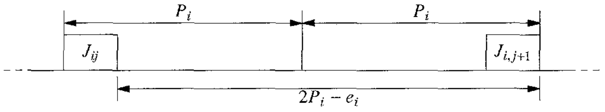


Fig. 2. Longest distance between two consecutive jobs of a periodic task.

Since the pinwheel problem can be viewed as a special case of the DC scheduling problem, the schedulers designed for the pinwheel problem, e.g., **Sa**, **Sx**, **Sbc**, **Sby**, and **Sxy**, can also be used to schedule DC task sets. Before using these schedulers, we first need to transform a DCTS instance into a pinwheel instance. The transformation can be done as follows. Each task T_i is transformed into an element, a_i, in the pinwheel instance, where $a_i = \lfloor c_i / \lceil e_i \rceil \rfloor$. Every $\lceil e_i \rceil$ consecutive time slots allocated to the ith symbol of the pinwheel instance are actually allocated to one job request of task T_i. In this way, we can use the algorithms designed for the pinwheel problem to solve the DCTS problem. Note that the distance constraints are satisfied since $\lceil e_i \rceil \cdot a_i = \lceil e_i \rceil \cdot \lfloor c_i / \lceil e_i \rceil \rfloor \le c_i$. For example, if task T_i has an execution time $e_i = 3.2$ and a distance constraint $c_i = 9.5$, its corresponding pinwheel instance a_i is $\lfloor 9.5 / \lceil 3.2 \rceil \rfloor = 2$. Suppose we can find for the transformed pinwheel instance a feasible schedule in which two consecutive slots allocated to the ith symbol of the pinwheel instance have a distance less than or equal to 2. If we group every $\lceil 3.2 \rceil = 4$ consecutive slots allocated to the ith symbol and treat them as the slots allocated to each job of task T_i (Fig. 3), then the distance between two consecutive jobs of task T_i will be less than or equal to $4 \cdot 2 = 8$, i.e., the distance constraint $c_i = 9.5$ of task T_i is satisfied. As mentioned earlier, the best known density threshold for scheduling pinwheel instances is 0.7 (Scheduler **Sxy**) [16]. Hence, as long as $\sum_{i=1}^{n} 1/\left(\left\lfloor c_i/\lceil e_i \rceil \right\rfloor\right) \le 0.7$, the DC task set is guaranteed to be schedulable. The major drawback of this approach is that it is not efficient in terms of density threshold (note the ceiling and floor functions in the schedulability condition $\sum_{i=1}^{n} 1/\left(\left\lfloor c_i/\lceil e_i \rceil \right\rfloor\right) \le 0.7$) and in terms of implementation (note that there may be $\lceil e_i \rceil$ preemptions for each task T_i).

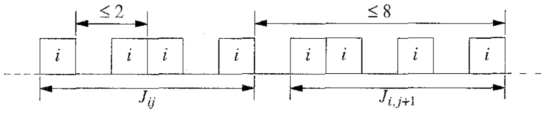


Fig. 3. A schedule for task T_i with $e_i = 3.2$ and $c_i = 9.5$.

With a careful analysis, we extend Scheduler **Sx** into a new scheduling algorithm for scheduling DC task sets. The proposed algorithm has a better schedulability condition, and can be implemented as easily as the RM algorithm for the periodic task systems (since the proposed algorithm is based on the RM algorithm). We present the algorithm and its associated results in the next section.

4 Distance-Constrained Scheduling Algorithm Based on Sx

In this section, we first briefly describe Scheduler **Sx** based on which we design our scheduling algorithm, Scheduler **Sr**, for DC task sets. We then elaborate on the design of **Sr**.

4.1 Scheduler Sx

Let $\mathbf{A} = \{a_1, a_2, \dots, a_n\}$ be an instance of the pinwheel problem. Without loss of generality, we assume in the following discussion that $a_1 \le a_2 \le \cdots \le a_n$. In [9], [17], it has been proved that if a pinwheel instance **A** with a total density ≤ 1 *consists solely of multiples* (i.e., $\rho(\mathbf{A}) = \sum_{i=1}^{n} 1/a_i \le 1$, and $a_i \mid a_j$ for all $i < j$, where $x \mid y$ denotes x divides y), then **A** is schedulable. For example, {2, 4, 8, 8} is schedulable. Based on this result, Scheduler **Sx** first transforms an arbitrary instance **A** to another instance $\mathbf{B} = \{b_1, b_2, \dots, b_n\}$ which consists solely of multiples and $b_i \le a_i$ for all i. (The detailed operation of the transformation will be discussed later in this section.) Since $b_i \le a_i$ for all i, the transformed multiset $\mathbf{B} = \{b_1, b_2, \dots, b_n\}$ is more restricted than the original pinwheel instance **A**. If we can find a pinwheel schedule for **B**, in which two consecutive occurrences of symbol "i" are scheduled with a distance less than or equal to b_i, the schedule also satisfies the original constraint a_i. Also, since the instance **B** consists solely of multiples (i.e., if $i < j$ then $b_i \mid b_j$), **B** can be feasibly scheduled if and only if $\rho(\mathbf{B}) \le 1$, as mentioned earlier.

Since $b_i \le a_i$ for all i, we have $\rho(\mathbf{B}) \ge \rho(\mathbf{A})$. If the total density of **A** is larger than 1, it is impossible to find a feasible schedule for **A**, i.e., the total density less than or equal to 1 is a *necessary* condition for a pinwheel instance to be schedulable. The *density threshold* ρ^* is then derived in such a way that as long as the total density of **A** is less than or equal to ρ^* then $\rho(\mathbf{B}) \le 1$ (i.e., **B** is schedulable). In other words, with the transformation of the problem instance, one can schedule all pinwheel instances with total density ≤ ρ^*. Note, however, that if a pinwheel instance **A** has a total density greater than ρ^*, it does not necessarily imply that the instance is not schedulable by the above method. **A** can be feasibly scheduled as long as the total density of the transformed multiset **B** is less than or equal to 1.

The operation used to transform an arbitrary pinwheel instance to one which consists solely of multiples is described as follows. Given an integer a, find a b_i for each a_i that satisfies

$$b_i = a \cdot 2^j \le a_i < a \cdot 2^{j+1} = 2b_i,$$

for some integer $j \ge 0$, i.e., $b_i = a \cdot 2^{\lfloor \log(a_i/a) \rfloor}$ (note that all logarithms in this paper are to the base 2). This operation is called *specializing* **A** *with respect to* $\{a\}$ in [10]. Note that the specialization operation can be done in $O(n)$ time assuming that, given a number a, the value $b_i = a \cdot 2^{\lfloor \log(a_i/a) \rfloor}$ can be computed in constant time. Fig. 4 illustrates the rationale behind the specialization operation. As shown in Fig. 4, all numbers in the pinwheel instance that fall in the range $[a \cdot 2^j, a \cdot 2^{j+1})$ are specialized (transformed) to $a \cdot 2^j$, for $j \ge 0$.

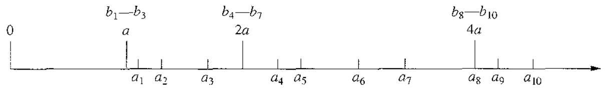


Fig. 4. Illustration of the rationale behind the specialization operation.

Scheduler **Sx** works as follows [10]. **Sx** first tries to find an integer x, $a_1/2 < x \le a_1$, and specializes **A** with respect to $\{x\}$ to get the specialized multiset **B**. Starting from $x = a_1$ down to $x = a_1/2 + 1$, **Sx** specializes **A** with respect to $\{x\}$ and chooses an x that minimizes $\rho(\mathbf{B})$, or chooses the first x which makes $\rho(\mathbf{B}) \le 1$ (or it finds that no such integer exists). If the total density of **B** is less than or equal to 1, **Sx** then uses the scheduling algorithm **SpecialSingle** [10] to find a feasible schedule for **B**. Since $b_i \le a_i$ for all i, the schedule found for **B** is also a feasible schedule for **A**. For notational convenience, we use **Sx** to refer to either the scheduler (the combination of the specialization operation and the scheduling algorithm) or just the specialization operation itself. For example, if $\mathbf{A} = \{4, 6, 7, 13, 24, 28, 33\}$ is specialized with respect to {4}, the specialized multiset is $\mathbf{B} = \{4, 4, 4, 8, 16, 16, 32\}$ with a total density of $33/32 > 1$, and if **A** is specialized with respect to {3}, the specialized multiset is $\mathbf{B} = \{3, 6, 6, 12, 24, 24, 24\}$ with a total density of $7/8$. Therefore, **Sx** will choose $x = 3$ and get $\mathbf{B} = \{3, 6, 6, 12, 24, 24, 24\}$.

It is shown in [10] that the density threshold for Scheduler **Sx** is $13/20$. That is, as long as the total density of **A** is less than or equal to $13/20$, the total density of the specialized multiset **B** will be less than or equal to 1, implying that **B** and also the original multiset **A** are schedulable.

By extending **Sx**, we design an efficient algorithm **Sr** to schedule general DC task sets. Specifically, let **T** = $\{T_1, T_2, \ldots, T_n\}$ be a DC task set in which T_i has an execution time e_i and a distance constraint c_i, for $1 \le i \le n$. (Without loss of generality, we assume $c_1 \le c_2 \le \cdots \le c_n$.) We will devise a polynomial time algorithm which can find a real number r, $c_1/2 < r \le c_1$, for a given DC task set **T**, so that **T** has a minimum density increase when its distance constraint multiset **C** = $\{c_1, c_2, \ldots, c_n\}$ is specialized with respect to $\{r\}$ to get **B** = $\{b_1, b_2, \ldots, b_n\}$ (note that $b_i \mid b_j$ for all $i < j$). We can then use **B**, instead of **C**, as the distance constraint multiset of **T**. In what follows, we first propose in Section 4.2 a scheduling algorithm, called *distance constraint-monotonic* (DCM) algorithm, which finds a feasible schedule for a DC task set whose distance constraint multiset consists solely of multiples and whose total density is less than or equal to 1. We then elaborate on the specialization operation in Section 4.3.

4.2 The DCM Algorithm

As mentioned in Section 2.2, there are two types of priority-driven preemptive scheduling schemes commonly used for scheduling traditional periodic task systems: the fixed-priority scheme and the dynamic-priority scheme. The proposed DCM algorithm is a fixed-priority preemptive scheduling scheme for scheduling DC task sets.

Similar to the RM algorithm, the DCM algorithm assigns priorities to tasks before run time in such a way that tasks with smaller distance constraints get higher priorities (ties are broken arbitrarily). At run-time, the system always executes the task with the highest priority among all active tasks. Since every job is ready to be executed as soon as its predecessor job is finished, if we do not add additional constraints, we may end up executing the highest-priority task over and over again but no others. In order for other lower-priority jobs to be executed, we need to artificially delay the execution of a higher-priority job. To this end, in addition to assigning a fixed priority to each task T_i, we also assign a (fixed) *separation constraint* [18], s_i, to each task T_i. The separation constraint s_i requires that any two consecutive jobs of task T_i be separated by at least s_i. Formally, if job $J_{i,j-1}$ is finished at time $f_{i,j-1}$ then the next job J_{ij} cannot start its execution before time $r_{ij} = f_{i,j-1} + s_i$. That is, s_i can be viewed as a relative ready time for each job of T_i. The ready time of a job is relative to the finishing time of the predecessor job of the same task, not to time 0. As shown in Fig. 5, the feasible execution interval for a distance-constrained job J_{ij} is $[f_{i,j-1} + s_i, f_{i,j-1} + c_i]$, where $f_{i,j-1}$ is the finishing time of $J_{i,j-1}$ and is not defined until run-time. (In contrast, the feasible execution interval for a periodic job J_{ij} is $[(j-1) \cdot P_i, j \cdot P_i]$, which is known before run-time.)

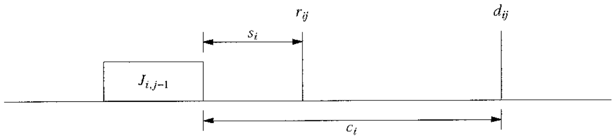


Fig. 5. The feasible execution interval $[r_{ij}, d_{ij}]$ of J_{ij}.

In Theorem 1, we show that, if $c_i \mid c_j$ for all $i < j$, the DCM algorithm can guarantee to find a feasible schedule for a task set with a total density less than or equal to 1 by setting $s_i = c_i - f_{i1}$, where f_{i1} is the finishing time of the first job of T_i. Note that we assume that the first job J_{i1} of T_i, $1 \le i \le n$, is ready at time 0 and must be finished by time c_i. In summary, the DCM algorithm is a priority-driven preemptive scheduling algorithm with the distance constraint-monotonic priority assignment and the above separation constraint assignment ($s_i = c_i - f_{i1}$, for $1 \le i \le n$).

THEOREM 1. *For a DC task set* **T**, *if* $c_i \mid c_j$ *for all* $i < j$ *and* $\rho(\mathbf{T}) \le 1$, *and if the separation constraint* $s_i = c_i - f_{i1}$ *is imposed on each task* T_i, $1 \le i \le n$, *the task set can be feasibly scheduled using the distance constraint-monotonic (DCM) priority assignment.*

PROOF. Similar to the proofs in [10] and [9], we need to show that

1) J_{i1} is finished before time c_i, i.e., $f_{i1} \le c_i$, for all i, and
2) for any task T_i, if it is running at time t, then it is also running at time $t + q \cdot c_i$ for any integer q such that $t + q \cdot c_i \ge 0$.

We prove 1) and 2) by induction on task id i. For task T_1, since it is the highest-priority task, its first job J_{11} will be running without preemption from time 0 to time e_1. Hence, $f_{11} = e_1 \le c_1$ and 1) is satisfied for T_1 (note that if $e_1 > c_1$ then the total density will be greater than 1, i.e., $\rho(\mathbf{T}) > 1$). Moreover, its separation constraint s_1 is set to $c_1 - e_1$. This separation constraint will make all later jobs of T_1 to be executed from time $q \cdot (e_1 + s_1) = q \cdot c_1$ to time $q \cdot c_1 + e_1$, for all $q \ge 1$. Therefore, 2) is satisfied for task T_1. Also notice that the distance constraints between J_{1j} and $J_{1,j+1}$, for $j \ge 1$, are all satisfied, i.e., $f_{i,j+1} - f_{ij} = c_i$.

Now, suppose 1) and 2) hold for all $i < k$. Let S_{ij} be the starting time of J_{ij}. It is easy to see that $f_{i,j+1} - f_{ij} = S_{i,j+1} - S_{ij} = c_i$, for all $1 \le i < k$ and all $j \ge 1$. Since

i) $c_i \mid c_k$, for all $1 \le i < k$,
ii) $f_{i1} \le c_i$, for all $1 \le i < k$, by the induction hypothesis of 1), and
iii) $f_{i,j+1} - f_{ij} = S_{i,j+1} - S_{ij} = c_i$,

we know that the amount of processor time allocated to T_i, for $1 \le i \le k$, from time 0 to time c_k is exactly $e_i \frac{c_k}{c_i}$.

If 1) is not true for T_k, we have

$$\sum_{i=1}^{k-1} e_i \frac{c_k}{c_i} + e_k = \sum_{i=1}^{k} e_i \frac{c_k}{c_i} > c_k,$$

or

$$\sum_{i=1}^{k} \frac{e_i}{c_i} > 1,$$

a contradiction to the assumption that the total density of the task set is less than or equal to 1.

Now, since J_{k1} is finished at time $f_{k1} \le c_k$, its separation constraint s_k is set to $c_k - f_{k1}$, which means that its successor job J_{k2} is ready at time c_k. Because $c_i \mid c_k$, for $1 \le i < k$, by the induction hypothesis of 2), the execution pattern of tasks $T_1, T_2, \ldots, T_{k-1}$ in the time interval $[0, c_k]$ repeats in the time intervals $[(j-1) \cdot c_k, j \cdot c_k]$, for all $j > 1$. Using the separation constraint assignment, it is easy to see that the execution pattern of task T_k also repeats in each of the time intervals $[(j-1) \cdot c_k, j \cdot c_k]$, for $j \geq 1$. Note that the lower-priority tasks $T_{k+1}, T_{k+2}, \ldots, T_n$ have no effect on the execution pattern of the higher-priority tasks $T_1, T_2, \ldots, T_k$. Hence, 2) is true for T_k.

From 1) and 2), we conclude that the schedule produced by the DCM algorithm, i.e., by the distance constraint-monotonic priority assignment and the separation constraint assignment defined in the theorem, is always feasible, i.e., each task T_i meets its distance constraint ($f_{i1} \leq c_i$ and $f_{i,j+1} - f_{ij} \leq c_i$, for all $j \geq 1$). □

EXAMPLE 1. Consider a set of distance-constrained tasks $\mathbf{T} = \{T_i = (e_i, c_i) \mid i = 1, 2, 3\} = \{(0.5, 3), (1, 6), (2.5, 12)\}$. The subschedule from time 0 to 12 produced by the DCM algorithm is shown in Fig. 6. The subschedule from time $q \cdot 12$ to $(q+1) \cdot 12$, for $q \geq 1$, is exactly the same as that from time 0 to 12. Note that $f_{i1} \leq c_i$ and $f_{i,j+1} - f_{ij} = c_i$, for $i = 1, 2, 3$, and $j \geq 1$.

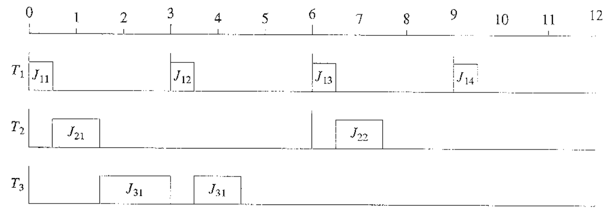


Fig. 6. The schedule produced by the DCM algorithm for the set of distance-constrained tasks in Example 1.

Note that if $c_i \mid c_j$ for all $i < j$, the schedule produced by the DCM algorithm is exactly the same as the one produced by the RM algorithm for the periodic task scheduling problem with task set $\mathbf{T} = \{T_i = (e_i, c_i) \mid 1 \leq i \leq n\}$, where c_i is the period of T_i. It is also worth mentioning that for a periodic task set whose period set consists solely of multiples, the schedule produced by the RM algorithm is exactly the same as that produced by the EDF algorithm. Moreover, as mentioned in Section 2.2, a periodic task set can be feasibly scheduled by the EDF algorithm if and only if its total utilization factor is less than or equal to 1. This gives another intuition of the correctness of the above theorem.

Liu and Layland [1] have shown that for periodic task systems, the largest response time (i.e., the largest difference between the request time and the finishing time of a job) for a task occurs when a job of the task is requested at the same time as the jobs of all higher-priority tasks. For the periodic task system in which the first jobs of all tasks are ready at time 0, the first job J_{i1} of task T_i has the largest response time among all jobs of T_i. The rationale behind the separation constraint assignment $s_i = c_i - f_{i1}$ is that if the first job J_{i1} of a distance-constrained task T_i has the largest response time among all jobs of T_i, then setting the separation constraint s_i to $c_i - f_{i1}$ can guarantee that all jobs of T_i meet their deadlines as long as the first job J_{i1} meets its deadline c_i (i.e., $f_{i1} \leq c_i$).

Although the above statement is true if the distance constraint multiset $\mathbf{C}$ consists solely of multiples, it is not true for general DC task sets. Therefore, for a general DC task set, if we do not perform the specialization operation before we schedule the task set (by the DCM algorithm), no density threshold exists. For example, the task set with $\{T_i = (e_i, c_i) \mid 1 \leq i \leq 6\} = \{(6, 59), (1, 87), (4, 167), (3, 204), (1, 422), (136, 4222)\}$ cannot be feasibly scheduled by the DCM algorithm without performing the specialization operation first, and its total density is only about 0.1864. If we first specialize the task set (the distance constraint multiset) with respect to $\{59\}$, we get the specialized task set $\{(6, 59), (1, 59), (4, 118), (3, 118), (1, 236), (136, 3776)\}$, with a density 0.2182. We can then feasibly schedule the specialized task set with the DCM algorithm. Moreover, the feasible schedule found for the specialized task set is also a feasible schedule for the original task set, because the distance constraint of each task in the original task set is looser than the corresponding distance constraint in the specialized task set. This example shows that performing the specialization operation before applying the DCM algorithm to a DC task set will, in general, result in a better performance. We will show in Section 4.4 that the density threshold for scheduling a DC task set using the specialization operation **Sr** discussed in Section 4.3 and the DCM algorithm is exactly the same as the utilization bound for scheduling periodic tasks using the RM algorithm.

4.3 Specialization Operation Sr

As discussed earlier, for a general DC task set, we can first specialize the task set $\mathbf{T}$ (the distance constraint multiset $\mathbf{C}$) with respect to a number (not necessarily an integer) and then find a feasible schedule for the specialized task set using the DCM algorithm as long as the total density of the specialized task set is less than or equal to 1. In this section, we discuss in details the specialization operation **Sr** that we use to specialize a general DC task set. Again, for notational convenience, we use **Sr** to refer to either the scheduler (the combination of the specialization operation and the DCM scheduling algorithm) or just the specialization operation itself.

Sr is a generalization of **Sx**. **Sx** specializes $\mathbf{A}$ with respect to $\{x\}$, where x is an integer chosen from the range $(a_1/2, a_1]$ so that the specialized task set has a minimum density increase (note that the density of the specialized task set is always larger than or equal to that of the original task set). Similarly, **Sr** specializes $\mathbf{C}$ with respect to $\{r\}$, where r is a real number chosen from the range $(c_1/2, c_1]$ so that the specialized task set has a minimum density increase. By generalizing the results in [10], we next present a polynomial time algorithm to find an r that minimizes the density increase. The specialization operation **Sr** uses this algorithm to find the best r and then specializes the distance constraint multiset $\mathbf{C}$ with respect to $\{r\}$.

The algorithm works as follows. Let $\mathbf{T}$ be a DC task set with distance constraint multiset $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$, where $c_1 \leq c_2 \leq \cdots \leq c_n$. Define $l_i = c_i \big/ 2^{\lceil \log(c_i/c_1) \rceil}$ (note that $c_1/2 < l_i \leq c_1$), for $1 \leq i \leq n$. Let $k_1 < k_2 < \cdots < k_u$, $u \leq n$, be the sorted sequence of l_is with duplicates removed. Since $l_1 = c_1$, we know that $k_u = c_1$. We call $\{k_1, k_2, \ldots, k_u\}$ the *special base* of $\mathbf{T}$.

For convenience of discussion, we define $k_0 = c_1 /2$. As will be shown in Lemma 1, the r value, denoted by r^*, that minimizes the density increase can always be found in the special base, i.e., $r^* = k_v$, for some $v \in \{1, 2, \ldots, u\}$. Let $\Phi_T(r)$ be the density of the task set T with its distance constraint multiset C specialized with respect to $\{r\}$, and let $\Phi^*_T = \Phi_T(r^*) = \min_{c_1/2<r\le c_1} \Phi_T(r)$. We can compute $\Phi_T(k_v)$, for all k_v in the special base of T, select the one that results in the minimum value of $\Phi_T(k_v)$, and use that k_v for r in the specialization operation. With this algorithm, the specialization operation Sr can be done in time $O(n \log n)$ (the $\log n$ factor comes from the sorting of the l_is).

The rationale behind the claim that only the numbers in the special base of **T** need to be considered in finding r^* is now elaborated on. Given any r, $c_1/2 < r \le c_1$, we define $\tau_r = \{T_i \in \mathbf{T} \mid c_i = r \cdot 2^j$, for some integer $j \ge 0\}$, and call it the *r-based subset* of **T**. It is easy to see that $\tau_r \ne \varnothing$ (hence, $\rho(\tau_r) \ne 0$) if and only if $r = k_v$, for $1 \le v \le u$. Recall that when **C** is specialized with respect to $\{r\}$, every c_i in **C** is specialized to a number $b_i = r \cdot 2^{\lfloor \log(c_i/r) \rfloor}$. The reason we define the r-based subsets of **T** as above is that for every task $T_i \in \tau_{k_v}$, $c_i = l_i \cdot 2^{\lceil \log(c_i/c_1) \rceil} = k_v \cdot 2^{\lfloor \log(c_i/k_v) \rfloor}$ (because $l_i = k_v$ and $\log (c_i/k_v) = \lfloor \log (c_i/k_v) \rfloor = \lceil \log (c_i/c_1) \rceil$ is an integer); as a result, if $r = k_v$, c_i will be specialized to itself, and hence, the density of T_i will not increase after specialization. This is also the reason we only need to choose r^* from the special base $\{k_1, k_2, \ldots, k_u\}$ of **T** (see Lemma 1).

Suppose **B** is the multiset resulting from specializing **C** with respect to $\{r\}$. For a task $T_i \in \tau_{k_v}$ (i.e., $c_i = k_v \cdot 2^j$, for some $1 \le v \le u$ and $j \ge 0$), there are two possible $\frac{c_i}{b_i}$ values due to the relationship between k_v and r:

a) If $r > k_v$, since $r/2 < k_v < r$, we have $b_i = r \cdot 2^{j-1}$ and $\frac{c_i}{b_i} = \frac{2k_v}{r}$.

b) If $r \le k_v$, since $r \le k_v < 2r$, we have $b_i = r \cdot 2^j$ and $\frac{c_i}{b_i} = \frac{k_v}{r}$.

From a) and b), we know that, if $k_{v-1} < r \le k_v$, for some v, $1 \le v \le u$, then

$$\Phi_T(r) = \sum_{i=1}^{v-1} \frac{2k_i}{r} \rho\left(\tau_{k_i}\right) + \sum_{i=v}^{u} \frac{k_i}{r} \rho\left(\tau_{k_i}\right). \qquad (4.1)$$

Moreover, we can prove the following lemma.

LEMMA 1. *If $k_{v-1} \le r < k_v$, for some v, $1 \le v \le u$, and $r \ne k_0$, then $\Phi_T(r) = \frac{k_v}{r} \Phi_T(k_v) - \rho(\tau_r)$.*

A proof of Lemma 1 is given in the Appendix.

Recall that $\rho(\tau_r) \ne 0$ if and only if $r = k_v$, for $1 \le v \le u$. Therefore, Lemma 1 implies that $\Phi_T(r) > \Phi_T(k_v)$, for all r, $k_{v-1} < r < k_v$ (see Fig. 7). Thus, to find an r^* so that

$$\Phi_T(r^*) = \Phi^*_T = \min_{c_1/2<r\le c_1} \Phi_T(r),$$

we only need to compute $\Phi_T(k_v)$, for all k_v in the special base of T. To summarize, we list the pseudocode of the specialization operation **Sr** in Fig. 8, and give the time complexity in Theorem 2.

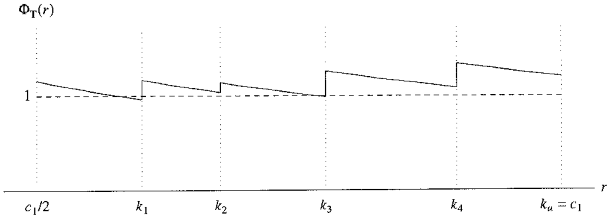


Fig. 7. The density function $\Phi_T(r)$.

```
/* Input: $\mathbf{T} = \{T_i = (e_i, c_i) \mid 1 \le i \le n\}$, where T is a DC task set
   and $c_i \le c_j$ for all $i < j$. */
/* Output: $r^*$, $\Phi_T(r^*)$, and $\mathbf{T}' = \{T'_i = (e_i, b_i \mid 1 \le i \le n\}$, where
   $b_i \mid b_j$ for all $i < j$. */
1. for $i$ := 1 to $n$ do $l_i = c_i / 2^{\lceil \log(c_i/c_1) \rceil}$;
2. sort $(l_1, l_2, \ldots, l_n)$ into nondecreasing order and remove
   duplicates;
   let $(k_1, k_2, \ldots, k_u)$ be the resulting sequence;
3. for $i$ := 1 to $n$ do put $T_i$ into subset $\tau_{l_i}$;
4. for $v$ := 1 to $u$ do $\rho(\tau_{k_v}) := \sum_{T_i \in \tau_{k_v}} e_i/c_i$;
5. compute $\Phi_T(k_u)$ according to (4.1);
6. for $v$ := $u - 1$ downto 1 do $\Phi_T(k_v) := \frac{k_{v+1}}{k_v} \Phi_T(k_{v+1}) - \rho(\tau_{k_v})$;
   /* see Lemma 1 */
7. find $r^*$ such that $\Phi_T(r^*) = \min_{r \in \{k_1, k_2, \ldots, k_u\}} \Phi_T(r)$;
8. for $i$ := 1 to $n$ do $b_i := r^* \cdot 2^{\lfloor \log(c_i/r^*) \rfloor}$;
9. output $r^*$, $\Phi_T(r^*)$, and $\mathbf{T}' = \{T'_i = (e_i, b_i) \mid 1 \le i \le n\}$.
```

Fig. 8. Specialization operation **Sr**.

THEOREM 2. *The time complexity of specialization operation* **Sr** *is $O(n \log n)$.*

PROOF. It is easy to see that all the steps in Fig. 8 can be done in $O(n)$ time, except step 2 (sorting), which can be done in $O(n \log n)$. □

Note that if we require that r^* must be an integer as in the pinwheel problem, our technique can still be used. We only need to compute $\Phi_T(k_u)$ according to (4.1) and then compute $\Phi_T(r)$ for r equal to $\lfloor k_u \rfloor$, $\lceil k_{u-1} \rceil$, k_{u-1}, $\lfloor k_{u-1} \rfloor$, …, $\lceil k_1 \rceil$, k_1, $\lfloor k_1 \rfloor$, one by one according to Lemma 1, and choose the r from $\{\lfloor k_i \rfloor, \lceil k_i \rceil \mid 1 \le i \le u - 1\} \cup \{\lfloor k_u \rfloor\}$ that minimizes $\Phi_T(r)$. This result implies that specialization operation **Sx** can also be done in $O(n)$ time by using (4.1) and Lemma 1.

4.4 Schedulability Condition of Sr

Scheduler **Sr** works as follows. Given a DC task set T, we first specialize **T** using the specialization operation **Sr** to get the specialized task set that has a minimum density increase and then use the DCM algorithm to schedule the specialized task set. In Theorem 3, we show that if

$$\rho(\mathbf{T}) \le K(n) = n(2^{1/n} - 1),$$

then specialization operation Sr can always find an r, $c_1/2 < r \le c_1$, such that $\Phi_{\mathbf{T}}(r) \le 1$. In other words, if $\rho(\mathbf{T}) \le n(2^{1/n} - 1)$, **T** is schedulable using Scheduler **Sr**. Before we prove Theorem3, we need the following two lemmas.

LEMMA 2. $k_{v+1}\Phi_{\mathbf{T}}(k_{v+1}) - k_v\Phi_{\mathbf{T}}(k_v) = k_v\rho(\tau_{k_v})$, *for* $1 \le v < u$.

A proof of Lemma 2 is given in the Appendix.

Let $U(d) = \{\mathbf{T} \mid \mathbf{T}$ is a DC task set with $\rho(\mathbf{T}) \le d\}$. To find the density threshold of Scheduler **Sr**, it suffices to find the maximum d such that $\Phi^*_{\mathbf{T}} \le 1$ for all $\mathbf{T} \in U(d)$.

LEMMA 3. *Let* $\mathbf{T} \in U(d)$ *such that, for all* $\mathbf{T}' \in U(d)$, $\Phi^*_{\mathbf{T}'} \le \Phi^*_{\mathbf{T}} = \Phi^*$, *and let* $\{k_1, k_2, \ldots, k_u\}$ *be the special base of* **T**. *It must be true that* $\Phi_{\mathbf{T}}(k_i) = \Phi^*$, *for all* k_i, $1 \le i \le u$.

A proof of Lemma 3 is given in the Appendix.

THEOREM 3. *Given a DC task set* **T**, *if* $\rho(\mathbf{T}) \le K(n) = n(2^{1/n} - 1)$, *then Scheduler* **Sr** *is guaranteed to find a feasible schedule for* **T**.

PROOF. Let $\mathbf{T}^* \in U(d)$ such that, for all $\mathbf{T}' \in U(d)$, $\Phi^*_{\mathbf{T}'} \le \Phi^*_{\mathbf{T}^*} = \Phi^*$. Let $\{k_1, k_2, \ldots, k_u\}$ be the special base of $\mathbf{T}^*$ and τ_r be the r-based subset of $\mathbf{T}^*$. From Lemma 3, we know that $\Phi_{\mathbf{T}^*}(k_i) = \Phi^*$, for all $1 \le i \le u$, and from Lemma 2, we have $k_{v+1}\Phi^* - k_v\Phi^* = k_v\rho(\tau_{k_v})$, for $1 \le v < u$. Therefore, for $1 \le v \le u - 1$,

$$\rho(\tau_{k_v}) = \frac{k_{v+1} - k_v}{k_v}\Phi^*. \tag{4.2}$$

Moreover, from (4.1) and (4.2), we have

$$\Phi^* = \Phi_{\mathbf{T}^*}(k_u) = \sum_{i=1}^{u-1}\frac{2k_i}{k_u}\rho(\tau_{k_i}) + \rho(\tau_{k_u})$$
$$= \frac{2}{k_u}(k_u - k_1)\Phi^* + \rho(\tau_{k_u}), \tag{4.3}$$

and hence,

$$\rho(\tau_{k_u}) = \frac{2k_1 - k_u}{k_u}\Phi^*. \tag{4.4}$$

Finally, from (4.2) and (4.4), we have

$$\rho(\mathbf{T}^*) = \sum_{i=1}^{u}\rho(\tau_{k_i}) = \left(\sum_{i=1}^{u-1}\frac{k_{i+1} - k_i}{k_i} + \frac{2k_1 - k_u}{k_u}\right)\Phi^*. \tag{4.5}$$

If $\mathbf{T}^*$ is schedulable by Scheduler **Sr**, Φ^* must be less than or equal to 1. If we set $\Phi^* = 1$ in (4.5), then (4.5) is exactly the same as (4) in [1] where the least upper bound of the processor utilization for the RM algorithm is derived (substituting u for m and $(k_u - k_i)/k_i$ for g_i in (4) of [1]). Moreover, the minimum value (over the k_is) of the term $\sum_{i=1}^{u-1}\frac{k_{i+1}-k_i}{k_i} + \frac{2k_1-k_u}{k_u}$ has been shown to be $K(u) = u(2^{1/u} - 1)$. Therefore, if we require that $\Phi^* \le 1$, we must have $\rho(\mathbf{T}^*) \le K(u) = u(2^{1/u} - 1)$. Since $u \le n$ and $u(2^{1/u} - 1) \ge n(2^{1/n} - 1)$, we conclude that, given a DC task set **T**, if $\rho(\mathbf{T}) \le K(n) = n(2^{1/n} - 1)$, then **T** is guaranteed to be schedulable by Scheduler **Sr**. □

Note that $\lim_{n\to\infty} n(2^{1/n} - 1) = \ln 2 \cong 0.69$. This density threshold is exactly the same as the utilization bound for scheduling periodic tasks using the well-known RM (rate-monotonic) algorithm [1]. This suggests that the two approaches can make the same kind of schedulability guarantee. In addition, we can also guarantee the temporal distance constraints using the DCTS model. Moreover, even if $\rho(\mathbf{T}) > n(2^{1/n} - 1)$, it does not necessarily mean that the task set **T** is not schedulable by Scheduler **Sr**. As long as the total density of the task set after specialization is less than or equal to 1, the task set can be feasibly scheduled by Scheduler **Sr**. To test if the total density of the task set after specialization is less than or equal to 1, we only need to perform the specialization operation, which can be done in $O(n \log n)$ time (Theorem 2). Therefore, the DCTS model is superior to the periodic task model in terms of the criterion for schedulability.

5 Applications of Distance-Constrained Scheduling Scheme

The DC scheduling scheme is useful both in real-time task scheduling and in real-time communications. In this section, we first describe how Scheduler **Sr** can be used to schedule a set of *sporadic* tasks (to be defined below), and then describe how **Sr** can be applied to real-time communications.

In Theorem 1, we have shown that given a DC task set **T** with $\rho(\mathbf{T}) \le 1$ and $c_i \mid c_j$ for all $i < j$, the distance constraints of all tasks can be satisfied by using the DCM algorithm (the distance constraint-monotonic priority assignment and the separation constraint assignment $s_i = c_i - f_{i1}$). Another important property of the DCM scheduling algorithm is stated in the following theorem.

THEOREM 4. *For a DC task set* **T**, *if* $c_i \mid c_j$ *for all* $i < j$ *and* $\rho(\mathbf{T}) \le 1$, *then the schedule produced by the DCM scheduling algorithm satisfies that during any time interval of length* c_i, *the total processor time allocated to task* T_i *is* e_i, *for all* i.

PROOF. In the proof of Theorem 1, we have shown that

1) J_{i1} is finished before time c_i, i.e., $f_{i1} \le c_i$, for all i, and
2) if a task T_i is running at time t, then it is also running at time $t + q \cdot c_i$ for any integer q such that $t + q \cdot c_i \ge 0$. (Recall that f_{ij} is the finishing time of J_{ij}.)

Thus, it is easy to see that in the time interval $[(j - 1) \cdot c_i, j \cdot c_i]$, the total processor time allocated to T_i is exactly e_i, for $1 \le i \le n$, and $j \ge 1$.

Now, to prove the theorem, we need to consider an arbitrary time interval $[t_1, t_2]$ of length c_i, where $0 \le t_1 \le j \cdot c_i \le t_2 \le (j + 1) \cdot c_i$, for some $j \ge 1$, and $t_2 - t_1 = c_i$. From 1) and 2), we know that the processor time allocated to T_i in the time interval $[j \cdot c_i, (j + 1) \cdot c_i]$ is exactly e_i, and from 2) we know that the processor allocation pattern in the time interval $[t_1, j \cdot c_i]$ repeats in the time interval $[t_1 + c_i, (j + 1) \cdot c_i] = [t_2, (j + 1) \cdot c_i]$. Therefore, it is obvious to see that the processor time allocated to T_i in the time interval $[t_1, t_2]$ is also e_i. □

Application to Sporadic Task Scheduling. Based on Theorem 4, Scheduler **Sr** can be used to schedule a set of *sporadic* tasks. In a sporadic task set $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$, each sporadic task T_i is characterized by its minimum interarrival time P_i, its (worst-case) execution time e_i, and its (relative) deadline D_i (we assume that $D_i \le P_i$). Similar to a periodic task, a sporadic task consists of an infinite sequence of job requests. However, the inter-arrival time between two consecutive jobs of a sporadic task is at least P_i. Specifically, if a job of sporadic task T_i arrives at time t, it needs to be executed for at most e_i units of time and must be completed by time $t + D_i$. Moreover, the next job of T_i will only arrive at time $t + P_i$ or later.

It is possible to schedule a sporadic task set using the *deadline-monotonic* (DM) algorithm [15] and perform the schedulability test by assuming the worst-case situation (i.e., treat the sporadic task set as a periodic task set with P_i as the period of T_i). However, in order to perform the DM priority-driven preemptive scheduling, this approach requires that the scheduler must be notified whenever a job of a sporadic task arrives. This requirement limits the application of this approach. Moreover, it is, in general, hard to derive a schedulability condition for the DM scheduling algorithm [15]. Although a schedulability condition for the case that $D_i = P_i$ is that as long as the total utilization (density) $\sum_{i=1}^{n} e_i/D_i$ is less than or equal to $n(2^{1/n} - 1)$, the (sporadic) task set is guaranteed to be schedulable by the DM (RM) algorithm, it is not known whether or not a task set is schedulable if the total utilization is larger than $n(2^{1/n} - 1)$ (but less than or equal to 1). By treating D_i as the distance constraint of T_i, Scheduler **Sr** can also be used to schedule sporadic task sets. From Theorem 4, we know that within any time interval of length $D_i' \le D_i$, **Sr** will allocate e_i units of time to T_i, as long as the total density $\sum_{i=1}^{n} e_i/D_i'$ of the task set after specialization is less than or equal to 1, where D_i' is the deadline (distance constraint) of T_i after the task set **T** is specialized by **Sr**. That is, as long as $\sum_{i=1}^{n} e_i/D_i' \le 1$, no matter when a job of T_i arrives at the system, there will be at least e_i processor time allocated to it before its (relative) deadline $D_i \ge D_i'$, and hence, its deadline can always be met. Moreover, from Theorem 3, we know that if $\sum_{i=1}^{n} e_i/D_i \le n(2^{1/n} - 1)$, then $\sum_{i=1}^{n} e_i/D_i' \le 1$. In other words, the system is more predictable by using Scheduler **Sr** than using the RM/DM scheduling algorithm.

Application to Real-Time Communications. Scheduler **Sr** can also be used to solve real-time communication problems. For interactive distributed services, such as multimedia conferencing and video/audio virtual realities, a certain amount of bandwidth must be allocated/reserved to deliver video/audio frames in time consistent with human perception and to provide users with a convenient means of guaranteeing message-transmission delay bounds.

A commonly-used message model in real-time communications is the *peak-rate* model [19], in which each message stream M_i is characterized by (C_i, D_i, P_i), where P_i is the minimum interarrival time between two successively-arrived messages, and C_i and D_i are the maximum message size and the relative deadline (relative to the arrival time) of the messages in the message stream M_i, respectively. A message stream with deadline constraint in real-time communications is similar to a sporadic task in real-time task scheduling. The difference is that message transmission is usually assumed to be nonpreemptive, while task scheduling is usually assumed to be preemptive. Although messages can be divided into a number of fixed-length smallest transmission entities, such as packets or cells, the transmission of a packet or a cell is still nonpreemptive. Therefore, real-time message transmission can be viewed as a discrete version of task scheduling in real-time task systems.

To show how **Sr** can be used to provide the timing guarantee for message transmission, we take the *isochronous* (real-time) service provided in a *Distributed Queue Dual Bus* (DQDB) MAN (metropolitan area network) as an example. As shown in Fig. 9, the DQDB network [20] consists of two high-speed (155 Mb/s) unidirectional *slotted* buses running in opposite directions to which every station is connected. Data transmission on both buses is slotted. Each slot can hold one 53-byte cell. Empty slots are generated by the slot generator at the head of each bus and transported "downstream." The isochronous service provided in DQDB requires that each real-time message stream is given a unique virtual circuit identifier (VCI). Empty *pre-arbitrated* (PA) slots are reserved/marked by the slot generator for all the real-time message streams with

1) a bit in the access control field (ACF) of each PA slot set to indicate the slot has been preassigned, and
2) the VCI field of each PA slot set to the appropriate VCI of the message stream to which the PA slot is preassigned.

The station with an isochronous message stream then watches for PA slots with the expected VCI and transmits its isochronous messages when those slots traverse through the station. The slot generator must ensure that the PA slots are properly allocated to each isochronous message stream M_i so that each message in M_i is transmitted within a time period $\le D_i$ after its arrival as long as the message interarrival time is $\ge P_i \ge D_i$ and the maximum message size (measured in cells) is $\le C_i$. In other words, the slot generator must assign at least C_i slots to message stream M_i between the arrival time and the deadline of *any* message of M_i.

It is easy to see that if a slot allocation scheme can allocate the PA slots in such a way that for *any* time interval of length D_i, at least C_i slots are allocated to M_i, for all i, then this allocation scheme can be used by the slot generator to allocate PA slots for real-time message streams in DQDB networks. This is exactly one of the properties of **Sr** proved in Theorem 4. As a result, we can first use the specialization

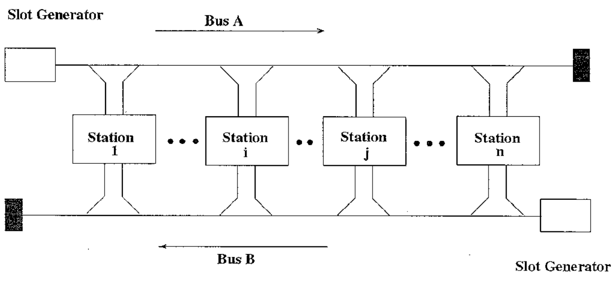


Fig. 9. The DQDB (IEEE802.6) network configuration.

operation to transform the deadline constraint multiset $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$ (assume $D_1 \leq D_2 \leq \cdots \leq D_n$) of a set of message streams $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \leq i \leq n\}$ to another deadline constraint multiset $\mathbf{D}' = \{D'_1, D'_2, \ldots, D'_n\}$ which consists solely of multiples and $D'_i \leq D_i$ for all i. That is, we find a D'_i for each D_i that satisfies $D'_i = D \cdot 2^j \leq D_i < D \cdot 2^{j+1} = 2D'_i$, for some integer $j \geq 0$, where D is an integer chosen from the range $(D_1/2, D_1]$ that results in the minimum total density

$$\rho(M') = \sum_{i=1}^{n} C_i / D'_i .$$

We then use a DCM-based on-line slot allocator to schedule the specialized message set, $\mathbf{M}' = \{M'_i = (C_i, D'_i) \mid 1 \leq i \leq n\}$, and generate a valid slot allocation schedule for the message stream set as long as $\rho(\mathbf{M}') \leq 1$. The technique used here is actually a discrete version of the technique used in scheduling sporadic real-time tasks. The interested reader is referred to [21] for a detailed account of how to apply Scheduler **Sr** to the DQDB isochronous service. With a similar approach, we can also apply Scheduler **Sr** to solve the token scheduling problem in a centralized-scheduling multiaccess LAN (local area network) [22] and many other real-time communication problems.

6 Conclusions

In this paper, we proposed the distance-constrained task system (DCTS) model which imposes that the temporal distance between any two consecutive executions of a task must be smaller than or equal to a predefined constraint. Since the DCTS model guarantees the temporal distance constraints which the periodic task model may not be able to effectively guarantee, it serves as a more intuitive and adequate scheduling model for "repetitive" task executions with temporal distance constraints.

We presented an efficient scheduling algorithm, Scheduler **Sr**, to schedule task sets with temporal distance constraints. We showed that a distance-constrained task set with n tasks can be feasibly scheduled by using Scheduler **Sr** as long as its total density is less than or equal to $n(2^{1/n} - 1)$. This provides the fundamental requirement of *predictability* in hard real-time applications. By predictability, we mean the deterministic guarantee that all tasks will meet their deadlines as long as the total density is held within the density threshold.

One interesting finding is that the derived density threshold $n(2^{1/n} - 1)$ is exactly the same as the well-known utilization bound derived in [1] for scheduling periodic task sets using the rate-monotonic scheduling algorithm. This suggests that the two approaches can make the same kind of schedulability guarantee. Moreover, as long as the total density of a DC task set after specialization is less than or equal to 1, the DC task set can be feasibly scheduled by Scheduler **Sr**. Therefore, the DCTS model is superior to the periodic task model in terms of the criterion for schedulability.

We also discuss how to apply the DCTS model and its scheduling scheme to the sporadic task scheduling problem and to the real-time communication problem in DQDB networks. We believe that the DCTS model can be applied to many real-time applications and provide better and more desirable solutions.

Appendix
Proofs of the Lemmas

Lemma 1. *If $k_{v-1} \leq r < k_v$, for some v, $1 \leq v \leq u$, and $r \neq k_0$, then* $\Phi_{\mathbf{T}}(r) = \frac{k_v}{r}\Phi_{\mathbf{T}}(k_v) - \rho(\tau_r)$.

Proof. From (4.1), we have, for $k_{v-1} < r < k_v$,

$$\begin{aligned}\Phi_{\mathbf{T}}(r) &= \sum_{i=1}^{v-1} \frac{2k_i}{r}\rho\left(\tau_{k_i}\right) + \sum_{i=v}^{u} \frac{k_i}{r}\rho\left(\tau_{k_i}\right)\\ &= \frac{k_v}{r}\left(\sum_{i=1}^{v-1} \frac{2k_i}{k_v}\rho\left(\tau_{k_i}\right) + \sum_{i=v}^{u} \frac{k_i}{k_v}\rho\left(\tau_{k_i}\right)\right)\\ &= \frac{k_v}{r}\Phi_{\mathbf{T}}\left(k_v\right)\end{aligned}$$

Note that $\rho(\tau_r) = 0$, for $k_{v-1} < r < k_v$.

For $r = k_{v-1}$, we have

$$\begin{aligned}\Phi_{\mathbf{T}}\left(k_{v-1}\right) &= \sum_{i=1}^{v-2} \frac{2k_i}{k_v - 1}\rho\left(\tau_{k_i}\right) + \sum_{i=v-1}^{u} \frac{k_i}{k_v - 1}\rho\left(\tau_{k_i}\right)\\ &= \sum_{i=1}^{v-1} \frac{2k_i}{k_v - 1}\rho\left(\tau_{k_i}\right) + \sum_{i=v}^{u} \frac{k_i}{k_v - 1}\rho\left(\tau_{k_i}\right) - \rho\left(\tau_{k_{v-1}}\right)\\ &= \frac{k_v}{v_{v-1}}\left(\sum_{i=1}^{v-1} \frac{2k_i}{k_v}\rho\left(\tau_{k_i}\right) + \sum_{i=v}^{u} \frac{k_i}{k_v}\rho\left(\tau_{k_i}\right)\right) - \rho\left(\tau_{k_{v-1}}\right)\\ &= \frac{k_v}{k_v - 1}\Phi_{\mathbf{T}}\left(k_v\right) - \rho\left(\tau_{k_{v-1}}\right)\end{aligned}$$

□

Lemma 2. $k_{v+1}\Phi_{\mathbf{T}}(k_{v+1}) - k_v\Phi_{\mathbf{T}}(k_v) = k_v\rho(\tau_{k_v})$, *for $1 \leq v < u$.*

Proof. From Lemma 1, for $1 \leq v < u$,

$$\Phi_{\mathbf{T}}(k_v) = \frac{k_{v+1}}{k_v}\Phi_{\mathbf{T}}(k_{v+1}) - \rho(\tau_{k_v}).$$

Therefore, $k_{v+1}\Phi_{\mathbf{T}}(k_{v+1}) - k_v\Phi_{\mathbf{T}}(k_v) = k_v\rho(\tau_{k_v})$. □

Lemma 3. *Let $\mathbf{T} \in U(d)$ such that, for all $\mathbf{T}' \in U(d)$,*

$$\Phi^*_{\mathbf{T}'} \leq \Phi^*_{\mathbf{T}} = \Phi^*,$$

and let $\{k_1, k_2, \ldots, k_u\}$ be the special base of $\mathbf{T}$. It must be true that $\Phi_{\mathbf{T}}(k_i) = \Phi^*$, for all k_i, $1 \leq i \leq u$.

PROOF. The lemma is trivially true for $u = 1$. We prove the lemma for $u > 1$ by contradiction. Suppose to the contrary that there exists a p, $1 \le p \le u$, $\Phi_T(k_p) > \Phi^*$. We can show that there exists a task set T′ with the same total density and the same special base as T, but $\Phi^*_{T'} > \Phi^*$, which contradicts the definition of **T**. To show this, we need to consider two cases:

Case 1: $1 < p \le u$.

Let's change the densities of τ_{k_p} and $\tau_{k_{p-1}}$ in **T** by changing the execution times of some of the tasks to create another task set **T′**:

a) τ_{k_p} is changed so that $\rho(\tau_{k_p})$ is increased by $\delta = \min\{\Phi_T(k_p) - \Phi^*, (1-\varepsilon)\rho(\tau_{k_{p-1}})\}$, and

b) $\tau_{k_{p-1}}$ is changed so that $\rho(\tau_{k_{p-1}})$ is decreased by δ,

where ε is a small positive number. Note that $\rho(\mathbf{T}) = \rho(\mathbf{T}')$, hence $\mathrm{T}' \in U(d)$. From (4.1), we know that for $k_{v-1} < r \le k_v$ and $1 \le v \le u$,

$$\Phi_T(r) = \sum_{i=1}^{v-1} \frac{2k_i}{r} \rho(\tau_{k_i}) + \sum_{i=v}^{u} \frac{k_i}{r} \rho(\tau_{k_i}).$$

Therefore, we have:

i) if $r \le k_{p-1}$, then $\Phi_{T'}(r) - \Phi_T(r) = \delta(k_p - k_{p-1})/r > 0$, i.e., $\Phi_{T'}(r) > \Phi^*$;

ii) if $r = k_p$, then

$$\Phi_{T'}(k_p) - \Phi_T(k_p) = \delta(k_p - 2k_{p-1})/k_p > -\delta \ge \Phi^* - \Phi_T(k_p),$$

i.e., $\Phi_{T'}(r) > \Phi^*$;

iii)if $r > k_p$, then $\Phi_{T'}(r) - \Phi_T(r) = 2\delta(k_p - k_{p-1})/r > 0$, i.e., $\Phi_{T'}(r) > \Phi^*$.

So, $\Phi^*_{T'} > \Phi^*_T$, which contradicts the definition of T in the lemma.

Case 2: $p = 1$.

In this case, we change the densities of τ_{k_1} and τ_{k_u} to create task set **T′**:

a) τ_{k_1} is changed so that $\rho(\tau_{k_1})$ is increased by $\delta = \min\{\Phi_T(k_1) - \Phi^*, (1-\varepsilon)\rho(\tau_{k_u})\}$, and

b) τ_{k_u} is changed so that $\rho(\tau_{k_u})$ is decreased by δ.

Again, we have $\rho(\mathbf{T}) = \rho(\mathbf{T}')$, hence $\mathbf{T}' \in U(d)$, and we have:

i) if $r = k_1$, then

$$\Phi_{T'}(k_1) - \Phi_T(k_1) = \delta(k_1 - k_u)/k_1 > -\delta \ge \Phi^* - \Phi_T(k_1),$$

i.e., $\Phi_{T'}(r) > \Phi^*$;

ii) if $r > k_1$, then $\Phi_{T'}(r) - \Phi_T(r) = \delta(2k_1 - k_u)/r > 0$, i.e., $\Phi_{T'}(r) > \Phi^*$.

So, again, $\Phi^*_{T'} > \Phi^*_T$, which contradicts the definition of **T** in the lemma. □

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments on an early draft of this paper.

REFERENCES

[1] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20, pp. 46-61, Jan. 1973.

[2] C.-C. Han and K.-J. Lin, "Scheduling Jobs with Temporal Consistency Constraints," *Proc. Sixth IEEE Workshop Real-Time Operating Systems and Software*, pp. 18-23, Pittsburgh, May 1989.

[3] C.-C. Han, K.-J. Lin, and J.W.-S. Liu, "Scheduling Jobs with Temporal Distance Constraints," *SIAM J. Computing*, vol. 24, pp. 1,104-1,121, Oct. 1995.

[4] A. Elsaadany, M. Singhal, and M. Liu, "Priority Communication Schemes on Local Area Networks for Multimedia Traffic," *Proc. 19th Conf. Local Computer Networks*, pp. 372-379, Oct. 1994.

[5] J. Boudec, "The Asynchronous Transfer Mode: A Tutorial," *Computer Networks and ISDN Systems*, vol. 24, pp. 279-309, 1992.

[6] M. Kawarasaki and B. Jabbari, "B-ISDN Architecture and Protocol, " *IEEE J. Selected Areas in Comm.*, vol. 9, pp. 1,405-1,415, Dec. 1991.

[7] T. Ng and V. Patel, "Timely Failure Detection in a Large Distributed Real-Time System," *Proc. Int'l Workshop Object Oriented Real-Time Dependable Systems*, Dana Point, Calif., 1994.

[8] "Coding of Moving Pictures and Associated Audio," SC29/WG11 Committee (MPEG) Draft of Standard ISO-IEC/JTC1 SC29, Nov. 1991.

[9] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The Pinwheel: A Real-Time Scheduling Problem," *Proc. 22nd Hawaii Int'l Conf. System Science*, pp. 693-702, Jan. 1989.

[10] M.Y. Chan and F. Chin, "Schedulers for Larger Classes of Pinwheel Instances," *Algorithmica*, vol. 9, pp. 425-462, 1993.

[11] S.-C. Cheng, J.A. Stankovic, and K. Ramamritham, "Scheduling Algorithms for Hard Real-Time Systems—A Brief Survey," *Tutorial Hard Real-Time Systems*, pp. 150-173. IEEE, 1988.

[12] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. Real-Time Systems Symp.*, pp. 166-171, Santa Monica, Calif., Dec. 1989.

[13] D.W. Leinbaugh, "Guaranteed Response Time in a Hard Real-Time Environment," *IEEE Trans. Software Eng.*, Jan. 1980.

[14] J.Y.-T. Leung and M.L. Merrill, "A Note on Preemptive Scheduling of Periodic, Real-Time Tasks," *Information Processing Letters*, vol. 11, pp. 115-118, Nov. 1980.

[15] J.Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation*, vol. 2, pp. 237-250, 1982.

[16] M.Y. Chan and F. Chin, "General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 755-768, June 1992.

[17] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel, "Pinwheel Scheduling with Two Distinct Numbers," *Theoretical Computer Science*, vol. 100, pp. 105-135, June 1992.

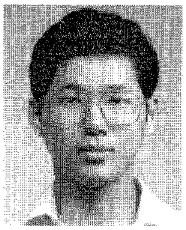
[18] C.-C. Han and K.-J. Lin, "Scheduling Real-Time Computations with Separation Constraints," *Information Processing Letters*, vol. 42, May 1992.

[19] C.M. Aras, J.F. Kurose,, D.S. Reeves, and H. Schulzrinne, "Real-Time Communication in Packet-Switched Networks," *Proc. IEEE*, vol. 82, pp. 122-139, Jan. 1994.

[20] I. For Electrical and E. Engineers, "IEEE Standards for Local and Metropolitan Area Networks: Distributed Queue Dual Bus (DQDB) Subnetwork of a Metropolitan Area Network (MAN)," IEEE 802.6, July 1991.

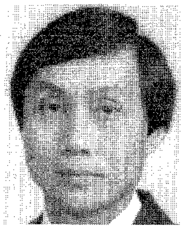
[21] C.-C. Han, C.-J. Hou, and K.G. Shin, "On Slot Allocation for Time-Constrained Messages in DQDB Networks," *Proc. IEEE INFOCOM '95*, pp. 1,164-1,171, Boston, Apr. 1995.

[22] C.-C. Han and K.G. Shin, "Real-Time Communication in Fieldbus Multiaccess Networks," *Proc. IEEE Real-Time Technology and Applications Symp.*, pp. 86-95, Chicago, May 1995.



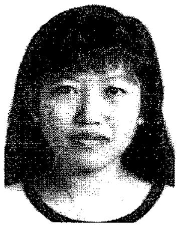
Ching-Chih Han (M'95) received the BS degree in electrical engineering from National Taiwan University in 1984, the MS degree in computer science from Purdue University in 1988, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 1992.

From August 1992 to January 1994, he was an associate professor in the Department of Applied Mathematics at National Sun Yat-sen University, Kaohsiung, Taiwan. He is now a visiting associate research scientist in the Real-Time Computing Laboratory at the University of Michigan, Ann Arbor. His current research interests include real-time communications, high-speed networking, real-time scheduling theory, distributed computing, and multimedia applications.



Kwei-Jay Lin received the BS degree in electrical engineering from the National Taiwan University, and the MS and PhD degrees in computer science from the University of Maryland, College Park. He was an associate professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign before moving to the Department of Electrical and Computer Engineering at the University of California, Irvine where he is now an associate professor. His research interests include real-time systems, scheduling theory, databases, and fault-tolerant systems.

Dr. Lin is a member of the IEEE Computer Society.



Chao-Ju Hou (S'88–M'94) received the BSE degree in electrical engineering in 1987 from National Taiwan University, the MSE degree in electrical engineering and computer science (EECS), the MSE degree in industrial and operations engineering, and the PhD degree in electrical engineering and computer science, all from the University of Michigan, Ann Arbor, in 1989, 1991, and 1993, respectively.

She is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Wisconsin, Madison. Her research interests are in the areas of distributed and fault–tolerant computing, real–time communications, queuing systems, estimation and decision theory, and performance modeling/evaluation. She is a recipient of Women in Science Initiative Awards, the University of Wisconsin-Madison. She is a member of the IEEE Computer Society, ACM Sigmetrics, and the Society of Woman Engineers.