Optimal Scheduling of Periodic Tasks
on Multiple Identical Processors *

Laura E. Jackson    George N. Rouskas
TR-98-14

December 23, 1998

Abstract

We consider the problem of scheduling $m$ periodic tasks on $n$, $n < m$, identical processors. Our main contribution is to show that the condition $\rho \leq n$, where $\rho$ is the total density of the task set, is a sufficient condition for scheduling the $m$ tasks such that no deadlines are ever missed. We start with a novel representation of the periodic task scheduling problem as a maximum network flow problem. The structure of the network ensures that task and processor conflicts are both avoided. Using results from network flow theory, we show that the maximum flow in the network is integer, and that there exists a feasible flow assignment in which all arc flows are integers. We also show that such a flow assignment corresponds to a feasible schedule for the original scheduling problem. Consequently, our work provides a positive answer to a question that has thus far remained open.

Keywords: Multiprocessor Scheduling, Periodic Tasks, Hard Real-Time Environment

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534

*This work was supported by a GAANN fellowship and the NSF under grant NCR-9701113.
1 Introduction

We consider the problem of scheduling $m$ periodic tasks on $n$ identical processors, where $n < m$. Each task $i$, $i = 1, \cdots, m$, is characterized by its computation time $C_i$ and deadline $D_i$, with $0 < C_i < D_i$. The tasks are periodic, i.e., each task requests new computation as soon as its current deadline expires, and thus, $D_i$ is also the period of the $i$-th task. We assume that all $C_i$ and $D_i$, $i = 1, \cdots, m$, are integers, and that time is slotted with a slot time equal to one unit of processing time. Task preemption is allowed, but only at slot boundaries.

The density of a task $i$ lies strictly between 0 and 1, and is defined to be:

$$\rho_i = \frac{C_i}{D_i}, \quad i = 1, \cdots, m$$  \hspace{1cm} (1)

The density of a set of $m$ tasks is defined as:

$$\rho = \sum_{i=1}^{m} \rho_i$$  \hspace{1cm} (2)

Obviously, the condition $\rho \leq n$ is a necessary condition for scheduling a set of tasks on $n$ processors. In the case of a single processor (i.e., $n = 1$), Liu and Layland [4] have shown that $\rho \leq 1$ is also a sufficient condition for schedulability. Two algorithms which can schedule a set of tasks with $\rho \leq 1$ on a single processor such that no task ever misses its deadline are the Earliest Deadline First (EDF) [4] and the Least Laxity [2] algorithms.

Unfortunately, it can be easily shown through counter-examples that the two algorithms above fail to produce feasible schedules in the multiprocessor case. Consider a set of $m$ tasks with a density $\rho \leq n$. Let $T = \gcd(D_1, \cdots, D_m)$ and $t = \gcd(T, T \times \frac{C_1}{D_1}, \cdots, T \times \frac{C_m}{D_m})$, where $\gcd$ stands for “greatest common divisor”. Dertouzos and Mok [2, Theorem 8] have shown that a sufficient condition for scheduling these $m$ tasks on $n$ processors is that $t$ be integral, and they presented an algorithm to schedule such a set of tasks. More recently, Khemka and Shyamasundar [3] have extended these results by deriving sufficient conditions for schedulability of task sets for which $t$ is not integral. Although Dertouzos and Mok conjecture that the condition $\rho \leq n$ is both necessary and sufficient for feasible scheduling in the multiprocessor case, to the best of our knowledge, showing that $\rho \leq n$ is sufficient for scheduling has remained an open problem.

In this paper we provide an answer to this open problem by proving that $\rho \leq n$ is indeed a sufficient condition for scheduling $m$ periodic tasks on $n$ identical processors. In Section 2 we formulate the problem, and in Section 3 we represent the periodic task scheduling problem as a maximum network flow problem. We then use results from network flow theory to show that the maximum flow in the network is integer, and that there exists a feasible flow assignment in which all arc flows are integers. Since this flow assignment corresponds to a feasible schedule for the original scheduling problem, we effectively prove that a feasible schedule exists as long as $\rho \leq n$. We then conclude the paper in Section 4.
2 Problem Formulation

We consider a set $S = \{1, \ldots, m\}$ of tasks and $n$ identical processors. Each task $i$ is specified by its computation time $C_i$ and period $D_i$, subject to $0 < C_i < D_i$. The density of the set $S$ of tasks is $\rho \leq n$. A schedule for this set of tasks is a one-to-many assignment of time slots to tasks that determines the tasks to be executed at a particular moment: if slot $\tau$ is assigned to task $i$, then task $i$ must be executed by some processor in slot $\tau$. In this work we consider preemptive schedules, but task preemption is allowed only at slot boundaries.

We will say that a schedule is feasible for a set $S$ of tasks, if it satisfies the following constraints:

- **Task constraint:** during a given slot, no task can be executed at more than one processor.
- **Processor constraint:** during a given slot, no processor may work on more than one task.
- **Deadline constraint:** no task ever misses any of its deadlines; in other words, each task $i$ receives exactly $C_i$ units of computation time within all consecutive intervals of length $D_i$.

The main contribution of our work is proving that a feasible schedule exists for a task set with density $\rho \leq n$.

If the density $\rho$ of a set $S$ of tasks is strictly less than $n$, we can add one or more “dummy” tasks to the set $S$ until the total density becomes equal to $n$. The scheduling of a dummy task on a processor $p$ during time slot $\tau$ corresponds to idle time for this processor in slot $\tau$. Therefore, without loss of generality we consider only problem instances for which

$$\rho = \sum_{i=1}^{m} \rho_i = n \quad (3)$$

Furthermore, we may also assume that the set $S$ of tasks is maximal in the sense that there does not exist a proper subset $S'$ of $S$ such that $\sum_{i \in S'} \rho_i = k$, for some integer $k < n$. If such a subset exists, then the original scheduling problem can be decomposed into two smaller instances: one involving finding a feasible schedule for the set $S'$ of tasks on $k$ processors, and the other involving finding a schedule for the set $(S - S')$ on $(n - k)$ processors.\footnote{1It is for the same reason that we require that the computational time $C_i$ of a task be strictly less than its period $D_i$; for if $C_i = D_i$ for some task $i$, we must assign a processor exclusively to task $i$, effectively reducing the original scheduling problem to one with $m - 1$ tasks and $n - 1$ processors.}

Let us define the schedule period $D$ as the least common multiple of all task periods: $D = \text{lcm}(D_1, \ldots, D_m)$. We note that we need only build a schedule for the first $D$ slots, i.e., from time 0 to time $D$. At time $D$ a new period begins for each task, just as at time 0. Therefore, if a feasible schedule exists for the task set $S$ on the interval $(0, D]$, by simply repeating this schedule at time $lD$, $l = 1, 2, \ldots$, we effectively construct a feasible schedule for $S$ for all times.

Within a schedule period $D$, there are $\frac{D}{D_i}$ periods of task $i$, and thus, the same number of instances of task $i$. Within each of these periods, the corresponding instance of task $i$ requires exactly $C_i$ units of work.
Given that we only consider task sets for which (3) holds, the minimum amount of work that must be done over a schedule period $D$ in order for all $m$ tasks to meet deadlines is:

$$\sum_{i=1}^{m} \frac{D}{D_i} C_i = \sum_{i=1}^{m} D \mu_i = nD$$

(4)

The maximum amount of work the processors are capable of completing in the same time interval is also $nD$. Consequently, a feasible schedule exists if and only if tasks can be assigned to the $nD$ available time slots in such a way that the task, processor, and deadline constraints are satisfied.

### 3 Network Representation

In this section we show that it is possible to construct a feasible schedule of length $D$ for a task set $S$ for which $\rho = n$. To this end, we represent an instance of the scheduling problem by a network, as follows. The set of nodes in the network consists of a source $s$, a sink $t$, and three other types of nodes. Nodes of the first type correspond to task instances. Since there are $m$ different tasks, and there are $D/D_i$ instances of task $i$ within a schedule period of length $D$, there are $\sum_{i=1}^{m} \frac{D}{D_i}$ of these nodes. There are $D$ nodes of the second type, each corresponding to one of the $D$ time slots of the schedule. Finally, for each time slot there are $n$ nodes of the third type, each representing one of the $n$ processors, for a total of $nD$ nodes of the third type.

There is an arc from the source node to each node corresponding to a task instance in the network. The capacity of the arc from the source to an instance of task $i$ is equal to the computation time $C_i$ required by the task. There is also an arc of unit capacity from each task instance to all time slots in the corresponding period of the task. For example, there is an arc from the node corresponding to the $k$-th instance, $k = 1, \cdots, \frac{D}{D_i}$, of task $i$ to each of the nodes corresponding to time slots $(k-1)D_i + 1, \cdots, kD_i$. From each time slot node, there is a unit capacity arc to each of the corresponding processor nodes. Finally, there is an arc of unit capacity from each of the $nD$ nodes corresponding to processors to the sink node.

The different types of nodes and arcs in the network are summarized in Tables 1 and 2, respectively. We note that the arc capacities are in units of work. Units of work flow out of the source, across the network, and out to the sink. By construction, the network will have a maximum flow of $nD$, the total amount of work that must be done to meet all the deadlines.

We now illustrate the network representation of the scheduling problem with an example. In Table 3 we show a problem instance with $n = 2$ processors, and $m = 3$ tasks, $a$, $b$ and $c$, with the given computation times, periods, and densities. The schedule period $D = \text{lcm}(3,4,4) = 12$ slots for this instance. The total density of this set of tasks is less than $n = 2$, so we add a fourth dummy task with period equal to $D = 12$ and computational time such that the total density becomes equal to $n$. The final task set is shown in Table 4. Note that for this set of four tasks, $T = \gcd(3,4,4,12) = 1$, and $t = \gcd(1, \frac{3}{4}, \frac{1}{4}, \frac{1}{4})$ is not integral. Therefore, these tasks do not satisfy the schedulability condition of [2]. On the other hand, this set of tasks does not satisfy either of the two schedulability conditions of [3] which are: (1) $\frac{C_i}{D_i} \leq (1 - \frac{1}{T})$ or $C_i = D_i$, for each task $i$, or (2) $\sum_{i=1}^{m} \left( T \times \frac{C_i}{D_i} - \left| T \times \frac{C_i}{D_i} \right| \right) \leq 1$. Thus, these tasks cannot be scheduled by the algorithms presented in [2, 3].
### Table 1: Types of Nodes

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source $s$</td>
<td>boldface square</td>
</tr>
<tr>
<td>Task instance</td>
<td>square</td>
</tr>
<tr>
<td>Time slot</td>
<td>circle</td>
</tr>
<tr>
<td>Processor</td>
<td>octagon</td>
</tr>
<tr>
<td>Sink $t$</td>
<td>boldface square</td>
</tr>
</tbody>
</table>

### Table 2: Types of Arcs

<table>
<thead>
<tr>
<th>Arc Type</th>
<th>Capacity</th>
<th>Initial Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source - Task instance</td>
<td>$C_i$</td>
<td>$C_i$</td>
</tr>
<tr>
<td>Task instance - Slot</td>
<td>1</td>
<td>$\rho_i$</td>
</tr>
<tr>
<td>Slot - Processor</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Processor - Sink</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: Task set with $m = 3$, $n = 2$, $\rho = \frac{17}{12} < n = 2$

<table>
<thead>
<tr>
<th>Job $i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$\rho_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>2</td>
<td>3</td>
<td>$\frac{8}{12}$</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>4</td>
<td>$\frac{3}{12}$</td>
</tr>
<tr>
<td>$c$</td>
<td>2</td>
<td>4</td>
<td>$\frac{6}{12}$</td>
</tr>
</tbody>
</table>

Table 4: Task set with $m = 4$, $n = 2$, $\rho = n = 2$

<table>
<thead>
<tr>
<th>Job $i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$\rho_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>2</td>
<td>3</td>
<td>$\frac{3}{12}$</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>4</td>
<td>$\frac{3}{12}$</td>
</tr>
<tr>
<td>$c$</td>
<td>2</td>
<td>4</td>
<td>$\frac{3}{12}$</td>
</tr>
<tr>
<td>$d$</td>
<td>7</td>
<td>12</td>
<td>$\frac{7}{12}$</td>
</tr>
</tbody>
</table>

We wish to build a schedule for the task set shown in Table 4 in the interval from time 0 to time $D = 12$. Figure 3 shows the network representation of this problem instance. The network is a directed graph in which all arcs point from left to right. From left to right, there are five partite sets of nodes, which we have drawn in the figure using the corresponding shapes listed in Table 1. The first set contains only the source node, labeled $s$. The second set of nodes represents task instances. Recall that a task instance is the manifestation of a task within a certain period. For example, consider task $a$ with period $D_a = 3$. On the time interval $[0,12]$, there are four instances of task $a$ (labeled $a_1$, $a_2$, $a_3$, and $a_4$, respectively, in Figure 3), each with a runtime requirement of $C_a = 2$. In addition to task $a$’s four instances, tasks $b$ and $c$ have three instances each, and task $d$ one in the first $D = 12$ slots, for a total of eleven nodes in the second partite set.

The third set of nodes corresponds to the $D = 12$ time slots. Specifically, the node labeled $k$, $k = 1, \cdots, 12$, represents the $k$-th time slot, i.e., the time interval $(k-1, k]$. The fourth set contains 24 processor nodes, corresponding to $n = 2$ processors (labeled $P$ and $Q$ in Figure 3) for each of the 12 time slots. The fifth and final set contains only the sink node $t$.

Four types of directed arcs connect the five partite sets from left to right, as summarized in Table 2. Each arc has unit capacity, with the exception of the arcs from the source node to the task instances, which have capacity equal to the corresponding instance’s computational time $C_i$. The arcs connecting the second and third set of nodes indicate the time slots in which a task instance can be executed. For example, there are three arcs leaving node $a_2$ and entering nodes 4, 5, and 6, in Figure 3. Since $a_2$ is the second instant of task $a$ with period $D_a = 3$, it must be executed in slots 4, 5, or 6. The arcs connecting the third and
fourth set of nodes indicate that task instances assigned to a certain slot will be executed in one of the two processors $P$ and $Q$.

Recall that a schedule is feasible if the Task, Processor, and Deadline constraints are satisfied. A network flow is called feasible if each arc flow is non-negative and does not exceed its capacity. A feasible network flow may not correspond to a feasible schedule. For example, assigning zero to each arc results in a feasible flow of value zero, but no work is done and every deadline is missed.

We now have the following result:

**Lemma 3.1** Consider a set $S$ of $m$ tasks, task $i$ associated with a computation time $C_i$ and period $D_i$ such that $0 < C_i < D_i$, and a total density $\rho = n$. Let $D = \text{lcm}(D_1, \ldots, D_m)$. A feasible flow in which all arc flows are integer is equivalent to a schedule in which the Task and Processor constraints are both satisfied.

**Proof.** By construction. Each time slot node has in-degree $m$ and is joined with exactly one instance of each of the $m$ tasks. For each $i$ and $\tau$, $i = 1, \ldots, m$, $\tau = 1, \ldots, D$, the arc from a task instance of task $i$ to time slot node $\tau$ has unit capacity and can therefore have a flow assignment of either 0 or 1. The $n$ outgoing arcs from $\tau$ each have unit capacity, so at most $n$ of the $m$ incoming arcs at node $\tau$ can have arc flow equal to 1. Each of those tasks can be (arbitrarily) paired with a processor (arbitrarily since processors are indistinguishable). That is, if task $i$ has arc flow of 1 into time slot node $\tau$ and is paired with processor $p$, then $p$ processes task $i$ during slot $\tau$. Each arc from node $\tau$ to processor $j$, $j = 1, \ldots, n$, has unit capacity, so at most one task will be processed by each processor within slot $\tau$. Thus, by construction, we have shown it possible to transform a feasible flow with integer arc flows into a schedule in which the Task and Processor constraints are both satisfied. □

Lemma 3.1 states that if we can find a feasible flow in which all arc flows are integer, then we can construct a schedule for the original periodic task scheduling problem that satisfies both the Task and Processor constraints. However, it does not answer the question of whether such a flow exists. The next lemma addresses this issue.

**Lemma 3.2** Consider the network representation of the periodic task scheduling problem. Then, there exists a feasible flow for which:

- all arc flows are integer, and

- each source-task instance arc has arc flow equal to its capacity.

**Proof.** A feasible flow can be obtained by making the initial assignment indicated in Table 2 (refer to Figure 3 for the initial flow assignment for the task set shown in Table 4). Each arc is assigned a flow equal to its capacity, with the exception of the task instance-time slot arcs, which are assigned a flow equal to the task's density ($\rho_i$). This flow assignment corresponds to relaxing the restriction that time is slotted, for it calls for a processor to spend $\rho_i < 1$ units of time on task $i$ during each slot. This feasible flow assignment has a flow value equal to $nD$ (in the example of Figure 3, $nD = 24$).
Figure 1: Network representation: Arc capacities for the task set of Table 4
Figure 2: Network representation: Initial flow assignment for the task set of Table 4
Let \( R \neq s \). The complement of \( R \) is \( \bar{R} \), and contains all nodes other than the source node \( s \). Then \([R, \bar{R}]\) is a source/sink cut of capacity \( nD \) (again, 24 in our example). The Max-Flow Min-Cut Theorem [1, Theorem 6.3] states that the maximum value of the flow from a source node \( s \) to a sink node \( t \) in a capacitated network equals the minimum capacity among all \( s-t \) cuts. By finding a cut and feasible flow for which the cut capacity equals the flow value, we have proven that \( nD \) is the maximum value of a feasible flow.

The Integrality Theorem [1, Theorem 6.5] states that if all arc capacities are integer, then the maximum flow is integer and, furthermore, there exists a feasible flow assignment in which each arc flow is also integer. Thus we have shown that there exists a flow assignment in which all arc flows are integer. Since the maximum flow is equal to \( nD \) and, by construction, the sum of the capacities of arcs from the source to the task instance nodes is also equal to \( nD \), then under this flow assignment each source-task instance arc is assigned a flow equal to its capacity, thus proving the second claim of the lemma.

We are now ready to state our main result. Recall that, if a task set satisfies \( \rho \leq n \), we can add “dummy” tasks to bring the total density to \( n \). Thus, Theorem 3.1 holds even when \( \rho < n \).

**Theorem 3.1** The condition \( \rho = n \) is sufficient for scheduling a set of \( m \) periodic tasks on \( n \) processors, \( 1 \leq n \leq m \).

**Proof.** Because of the first claim of Lemma 3.2, there exists a flow assignment of value \( nD \) such that all arc flows are integers. Because of Lemma 3.1, this assignment is equivalent to a schedule that satisfies the Task and Processor constraints. This schedule will also satisfy the Deadline constraints because: (a) the flow value is \( nD \), thus no processor is ever idle, and (b) the flow incoming to each task instance node (and, thus, the flow outgoing from such a node) is equal to the computation time of the task (second claim of Lemma 3.2), ensuring that each task instance receives processing equal to its requirements within its period. Thus, a feasible schedule exists when \( \rho = n \).

As a final note, the Ford-Fulkerson labeling algorithm [1] can be employed to find a flow assignment with a value of \( nD \) and for which all arc flows are integer. For example, a flow assignment for the task set of Table 4 with flow value 24 and all arc flows integer is shown in Figure 3, from which a feasible schedule can be easily constructed.

### 4 Concluding Remarks

We have considered the problem of multiprocessor scheduling of periodic tasks. We have developed a network representation of the scheduling problem, and, using results from network flow theory, we have provided a positive answer to the, thus far open, question of whether the condition \( \rho \leq n \) is sufficient for feasible scheduling on multiple processors.

While an algorithm exists for finding a feasible schedule, there are two practical problems associated with it. First, the algorithm requires advance knowledge of all task computation times and deadlines, and
All arc flows equal to arc capacity, except dotted arcs which have zero flow

Figure 3: Integer flow assignment for the task set of Table 4
second, its complexity is a function of the least common multiple of the task periods which can be quite large (e.g., when task periods are relatively prime). We are currently working on developing a computationally efficient algorithm that can be used for on-line scheduling of periodic tasks.
References


