A NOTE ON PREEMPTIVE SCHEDULING OF PERIODIC, REAL-TIME TASKS *

Joseph Y.-T. LEUNG and M.L. MERRILL
Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201, U.S.A.

Received 22 April 1980; revised version received 28 August 1980

Periodic real-time tasks, optimal scheduling algorithm, feasibility test, NP-completeness

1. Introduction

Computer systems are now being used for the control of a wide variety of real-time processes. In many of these real-time applications, the computer is required to execute programs in response to periodic, external signals and to guarantee that each such program will be completely processed before the occurrence of a subsequent signal. Sequencing tasks in such an environment raises a number of challenging theoretical questions whose solutions have much practical significance. In this note we shall address some of these issues and attempt to provide solutions for them.

A task system consists of a set of periodic, real-time tasks and is denoted by $(\{T_i\}, \{e_i\}, \{d_i\}, \{p_i\}, \{s_i\})$, where each T_i $(1 \leqslant i \leqslant n)$ is a periodic, real-time task consisting of a periodic sequence of requests for execution of a computation such that

(1) execution of the k^{th} $(k = 1, 2, ...)$ computation is requested at time $s_i + (k-1)p_i$ (p_i is the period of T_i),

(2) the k^{th} computation requires e_i units of execution time, and

(3) the k^{th} computation must be completed no later than the deadline $s_i + (k-1)p_i + d_i$, with $0 < e_i \leqslant d_i \leqslant p_i$.

A schedule S of a task system R is said to be *valid* if the deadlines of all task computations in R are met in S. A task system is said to be *feasible* on m (identical) processors if there exists a valid schedule for the task system on m processors. A scheduling algorithm is said to be *optimal* for m processors if it produces a valid schedule on m processors for every task system which is feasible on m processors.

At this point, two fundamental questions naturally arise:

(1) For all $m \geqslant 1$, are there optimal scheduling algorithms for m processors?

(2) What are the necessary and sufficient conditions for a task system to be feasible on m processors?

Labetoulle [2] and Liu and Layland [4] have shown that the Deadline Algorithm is optimal for a single processor. The Deadline Algorithm schedules, at each instant of time t, that active task [1] whose deadline is closest to t. Ties may be broken arbitrarily. At the present time, no algorithm has been found which is optimal for $m > 1$ processors.

Conditions which are both necessary and sufficient for a task system to be feasible on m processors are known only for a special type of task system [1,4]. If the task system is such that $d_i = p_i$ for all $1 \leqslant i \leqslant n$, then $\Sigma_{i=1}^{n} e_i/p_i \leqslant m$ is both a necessary and sufficient condition for a task system to be feasible on m processors [1,4]. For arbitrary task systems, the above condition is a necessary (but not sufficient) condition for the task system to be feasible on m processors. On the other hand, $\Sigma_{i=1}^{n} e_i/d_i \leqslant m$ is a sufficient (but not necessary) condition for a task system to be feasible on m processors. In this note we give evidence that there may not be any simple condition which is both

* This research was supported in part by the National Science Foundation under Grant MCS-79-04898.

[1] A task is said to be active if it has requested the execution of a computation, but has not yet received a processing time equal to its execution time.

necessary and sufficient for an arbitrary task system to be feasible on m processors. Specifically, we show that the problem of deciding whether an arbitrary task system is feasible on m processors is NP-hard for every $m \geq 1$. We also give an algorithm (which runs in exponential time) to decide if an arbitrary task system is feasible on a single processor.

2. Feasibility test

Since the Deadline Algorithm is optimal for a single processor, we can decide if a task system is feasible on a single processor by constructing a schedule using the Deadline Algorithm and checking to see if the schedule so constructed is valid. For this method to work, we need to establish an a priori time-bound within which we need to construct the schedule. If the task system is such that $s_i = s_j$ for all $1 \leq i, j \leq n$, then an obvious time-bound would be P = least common multiple of $\{p_1, ..., p_n\}$. However, if $s_i \neq s_j$ for some $1 \leq i, j \leq n$, then it is not clear that such a time-bound necessarily exists. We shall show that such a time-bound indeed exists and is given by s + 2P, where $s = \max\{s_1, ..., s_n\}$. Without loss of generality, we assume that $\min\{s_1, ..., s_n\} = 0$.

Before we present this result, we need to introduce the following notation. Let S be a schedule of the task system R. We define the configuration of the schedule S for the task system R at time t, denoted by $C_S(R, t)$, to be the n-tuple $(e_{1,t}, ..., e_{n,t})$, where $e_{i,t}$ is the amount of time for which task T_i has executed since its last request up until time t, and $e_{i,t}$ is undefined if $t < s_i$. Recall that the Deadline Algorithm schedules, at each instant of time t, that active task whose deadline is closest to t. In case of a tie, it may be broken by an arbitrary tie-breaking rule. Thus, schedules produced by the Deadline Algorithm are not unique. For convenience in the following discussion, we shall assume that the Deadline Algorithm resolves ties using a fixed tie-breaking rule so that it will produce a unique schedule for a given task system. The following two lemmas are instrumental in proving our result.

Lemma 1. Let S be the schedule of a task system R constructed by the Deadline Algorithm. Then for each task T_i and for each time instant $t_1 \geq s_i$, we have $e_{i,t_1} \geq e_{i,t_2}$, where $t_2 = t_1 + P$.

Proof. We proceed to prove the lemma by contradiction and assume that there is some task T_{j_1} and some time instant $t_1 \geq s_{j_1}$ such that $e_{j_1,t_1} < e_{j_1,t_2}$, where $t_2 = t_1 + P$. Then there must be some time $t_1' < t_1$ such that T_{j_1} is active at both t_1' and $t_2' = t_1' + P$, and T_{j_1} is executing at t_2' but not at t_1'. This can only occur if there is another task T_{j_2}, which is active at t_1' but not at t_2', such that the deadline of T_{j_2} is earlier than the deadline of T_{j_1}. But this means that $e_{j_2,t_1'} < e_{j_2,t_2'}$, and thus we may repeat the above argument to produce an infinite progression of task computations $T_{j_3}, T_{j_4}, ...$, for which no lower bound will exist for the time at which the task computations in the sequence are active. But this is impossible since every task T_i in the task system has an initial request time s_i.

Lemma 2. Let S be the schedule of a task system R constructed by the Deadline Algorithm. If R is feasible on one processor, then $C_S(R, t_1) = C_S(R, t_2)$, where $t_1 = s + P$ and $t_2 = s + 2P$.

Proof. If the lemma is not true, then, by Lemma 1, there must be a task T_{j_1} for which $e_{j_1,t_1} > e_{j_1,t_2}$. We first show that the processor is continuously executing task computations in the entire interval $[t_1, t_2]$. For if not, then there must be a time $t_1 + \Delta$ $(0 \leq \Delta \leq P)$ at which the processor is idle. But this implies that all task computations requested prior to $t_1 + \Delta$ have finished execution. By Lemma 1, we have $C_S(R, s + \Delta) = C_S(R, t_1 + \Delta)$. Since the task requests in the intervals $[s + \Delta, t_1 + \Delta]$ and $[t_1 + \Delta, t_2 + \Delta]$ are the same and since $C_S(R, s + \Delta) = C_S(R, t_1 + \Delta)$, the schedule in the intervals $[s + \Delta, t_1 + \Delta]$ and $[t_1 + \Delta, t_2 + \Delta]$ must be identical. But this means that $C_S(R, t_1) = C_S(R, t_2)$, contradicting our assumption that $C_S(R, t_1) \neq C_S(R, t_2)$.

Thus the processor is always busy in the interval $[t_1, t_2]$. Since $e_{i,t_1} \geq e_{i,t_2}$ for each $1 \leq i \leq n$ by Lemma 1, and since $e_{j_1,t_1} > e_{j_1,t_2}$ for some $1 \leq j_1 \leq n$, we may conclude that the amount of work requested by all tasks in R in the interval $[t_1, t_2]$ is strictly larger than $t_2 - t_1$. But this implies that $\Sigma_{i=1}^{n} e_i/p_i > 1$. Hence, R cannot be feasible on one processor.

Theorem 1. Let S be the schedule of a task system R constructed by the Deadline Algorithm. R is feasible on one processor if and only if (1) all deadlines in the interval $[0, t_2]$ are met in the schedule S, where $t_2 = s + 2P$, and (2) $C_S(R, t_1) = C_S(R, t_2)$, where $t_1 = s + P$.

Proof

(*only if part*). If R is feasible on one processor, then the schedule S constructed by the Deadline Algorithm must be a valid schedule. Thus, all deadlines in the interval $[0, t_2]$ are met in S. By Lemma 2, we have $C_S(R, t_1) = C_S(R, t_2)$.

(*if part*). For each nonnegative integer j, let us define t_j to be the time instant $s + jP$. For each $j \geqslant 2$, the requests made by all tasks in R in the interval $[t_j, t_{j+1}]$ must be identical to those made in the interval $[t_1, t_2]$. Since $C_S(R, t_1) = C_S(R, t_2)$, it is not difficult to see that the schedule S repeats itself every P units of time, starting from t_1. Since all deadlines in the interval $[0, t_2]$ are met in S, the deadlines of all task computations must also be met in S. Thus, R is feasible on one processor.

An algorithm to determine if a task system R is feasible on one processor consists of constructing a schedule S using the Deadline Algorithm until time $t_2 = s + 2P$. By Theorem 1, if all deadlines in $[0, t_2]$ are met in S and $C_S(R, t_1) = C_S(R, t_2)$, where $t_1 = t_2 - P$, then R is feasible. Otherwise, it is not feasible.

We now show that the above decision problem is NP-hard. We shall show that existence of a polynomial-time algorithm for solving this decision problem implies the existence of a polynomial-time algorithm for solving the following NP-complete, number-theoretic problem.

K Simultaneous Congruences Given n ordered pairs of positive integers $(a_1, b_1), ..., (a_n, b_n)$ and a positive integer K ($2 \leqslant K \leqslant n$), is there a subset of $\ell \geqslant K$ ordered pairs $(a_{i_1}, b_{i_1}), ..., (a_{i_\ell}, b_{i_\ell})$ such that there is a positive integer x with the property that $x \equiv a_{i_j} \bmod (b_{i_j})$ (i.e. $x = a_{i_j} + p_{i_j} * b_{i_j}$ for some positive integer p_{i_j}) for each $1 \leqslant j \leqslant \ell$?

The K Simultaneous Congruences problem has been shown to be NP-complete in [3].

Theorem 2. The problem of deciding whether a task system is feasible on one processor is NP-hard.

Proof. We show that a polynomial-time algorithm for our decision problem can be used to construct a polynomial-time algorithm for the K Simultaneous Congruences problem. Given an arbitrary instance of the K Simultaneous Congruences problem, $(a_1, b_1), ..., (a_n, b_n)$ and K, we construct a task system R consisting of n + 1 tasks as follows. $T_1, ..., T_n$ have the characteristics $e_i = 1/K$, $d_i = 1$, $p_i = b_i$, $s_i = a_i$ for all $1 \leqslant i \leqslant n$ and T_{n+1} has the characteristics $e_{n+1} = 1/K$, $d_{n+1} = 1$, $p_{n+1} = 1$ and $s_{n+1} = 0$. By our construction, each task computation in R requires 1/K unit of processing time and a deadline occurs 1 unit of time after each request; hence, for R to be feasible on one processor, no more than K task computations can be requested simultaneously. Since T_{n+1} requests at each integer unit of time and since each T_i ($1 \leqslant i \leqslant n$) requests only at integer-valued time, R is feasible on one processor if and only if no more than K − 1 tasks among $\{T_i\}_{i=1}^n$ request simultaneously. Thus, the instance of K Simultaneous Congruences problem has a solution if and only if R is *not* feasible on one processor. Since the construction can be carried out in polynomial time, a polynomial-time algorithm for our decision problem implies the K Simultaneous Congruences problem can be solved in polynomial time.

Corollary. The problem of deciding whether a task system is feasible on m processors is NP-hard for each $m \geqslant 1$.

Proof. In the above reduction, if $m > 1$, we can introduce m − 1 additional 'dummy' tasks all of which have execution times equal to the deadline and the period, say 1 unit, and the initial request times are 0. The rest of the proof follows immediately.

3. Discussions

We have given an algorithm to decide if an arbitrary task system is feasible on one processor. A natural extension of this work is to devise algorithms for deciding if an arbitrary task system is feasible on $m > 1$ processors. A closely related question is whether there are any optimal scheduling algorithms for $m > 1$ processors. It is conceivable that an optimal scheduling algorithm for $m > 1$ processors can be used in a similar way to construct an algorithm for deciding whether an arbitrary task system is feasible on $m > 1$ processors.

We have shown that the problem of deciding if an arbitrary task system is feasible on m processors is NP-hard for each $m \geqslant 1$. In [3] the K Simultaneous Congruences problem was only proved to be NP-complete in the 'ordinary' sense (i.e. it was not proved to be NP-complete in the 'strong' sense). It thus follows that our decision problem was only shown to be NP-hard in the ordinary sense and hence it does not preclude any possibility that the decision problem can be solved by a pseudo-polynomial time algorithm. On the other hand, the algorithm we gave clearly runs in more than pseudo-polynomial time in the worst case. It will be interesting to determine the exact boundary of the computational complexity of this problem.

Acknowledgment

We would like to thank the referee for suggesting the simplified proof of Lemma 2.

References

[1] E.G. Coffman, Jr., Introduction to deterministic scheduling theory, in: E.G. Coffman, Jr., Ed., Computer and Job-Shop Scheduling Theory (Wiley, New York, 1976) 1–50.

[2] J. Labetoulle, Some theorems on real time scheduling, in: E. Gelenbe and R. Mahl, Eds., Computer Architecture and Networks (North-Holland, Amsterdam, 1974) 285–293.

[3] J.Y.-T. Leung and J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, in preparation.

[4] C.L. Liu and J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61.