The Journal of Real-Time Systems, 2, 181–194 (1990)

Dynamic Scheduling of Real-Time Tasks under Precedence Constraints

H. CHETTO, M. SILLY AND T. BOUCHENTOUF
LAN (Laboratoire d'Automatique de Nantes), U.A. au CNRS n° 823, ENSM 1 Rue de la noé 44072 Nantes cédex 03 France

Abstract. While scheduling theory has been developed over a long period of time, it is important to note that most results concern problems with static characteristics. However, a real-time system is dynamic and requires on-line and adaptive scheduling strategies. So, an important aspect of real-time systems research is to devise methods flexible enough to react to a dynamic change of processor load and to attempt to schedule all the tasks judiciously.

In this paper, we are particularly concerned with the problem of scheduling tasks which are of two kinds: periodic and sporadic, on a monoprocessor machine. Periodic tasks are independent, run cyclically and their characteristics are known in advance. In addition, we allow for the unpredictable occurrence of aperiodic task groups, with timing and precedence constraints. Clearly, the main problem is to devise a schedulability test that makes it possible to decide whether a new occurring group can be accepted, without upsetting the tight timing behavior requirements.

We present an optimal acceptance test which returns a decision on the basis of the current state of processor load and by considering tasks to be scheduled according to the well known preemptive algorithm *Earliest Deadline*. The acceptance algorithm and a complexity analysis are presented.

1. Introduction

The objective of any real-time computing system design is to make certain that it can meet critical performance constraints imposed by the environment. In such a system, "correctness depends not only on the logical result of the computation but also on the time at which the results are produced (Stankovic 1988). More precisely, when all the tasks performed in the computing system must imperatively take place within a strict lapse of time after their activation (otherwise there might be severe consequences), then the system is said to be hard real-time. Examples of current hard real-time applications include flight control, nuclear power plants and automated manufacturing plants. In each of these systems, the time value is usually described in terms of deadlines by which tasks must be completed and the deadlines are traceable to the physical environment with which the computer(s) must interact.

In this paper, the time critical tasks of interest run on a single processor machine. This machine has its own private memory which is supplied with an operating system and the application software. It works in the feedback loop of the environment (the controlled system). Its goal is to derive inputs from sensors and then, its outputs are sent to control actuators or to update displays. To perform this control function which is initially well-defined, the computer has to run a set of tasks, cyclically in an indefinite loop. Values of task periods then depend on the dynamic of the physical process associated to them. Furthermore, the computing system has to cope with unpredictable changes in the environment whose effects can result in a sudden and temporary increase in processor workload.

This means that there exist additional tasks which lie dormant until they are activated. Although these so called sporadic tasks are initially passive and do not in that state require any processor time, they represent a latent demand for processor time that must be accounted for. For this reason, they can affect at a certain point in time, the scheduling of periodic tasks since they share the processor with them. Thus, an important topic in hard real-time systems research is the design of dynamic and adaptive schedulers with a view to meeting all the timing requirements during execution, whenever it is possible (Chetto and Chetto 1989).

Most of the research carried out so far in the area of real-time scheduling has been based on the assumption of a cyclical and static workload (Serlin 1972; Liu and Layland 1973). In (Zhao and Ramamritham 1985), a heuristic approach was described to perform on-line scheduling of periodic tasks in the presence of sporadic tasks with no inter-task constraints. When a sporadic task arrives on the machine and requires to be run, an acceptance routine is invoked and decides whether the task can be accepted. While this approach is efficient in terms of computational complexity, it is suboptimal since the test does not consider preemptions. A similar heuristic test was developed in (Cheng, Stankovic and Ramamritham 1986) for dynamic scheduling of groups of sporadic tasks with precedence constraints among themselves but did not consider periodic tasks. Recently, an optimal test was proposed in (Chetto and Chetto 1989) for scheduling periodic and independent sporadic tasks.

In this paper, we are specifically concerned with the problem of scheduling dependent sporadic tasks together with periodic tasks, preemptively. More precisely, the key problem amounts to determining on-line whether or not a group of sporadic tasks that need to be run on the machine can be accepted. Acceptance means that all the tasks of the group will be executed by their deadline and that all periodic and previously accepted sporadic tasks will still meet their deadlines. The accepted tasks are then queued at the local scheduler until they are selected to be processed. Otherwise, the resulting situation may be one of the following:

- rejection is taken into account by an operator which manually acts upon the controlled system;
- the rejected task group is sent to another machine. If we assume that the machine is connected to one or more through a communication network, a selection process will be performed in order to find the machine that will be able to handle the tasks. This problem has been dealt with in few papers (Ramamritham and Stankovic 1984; Cheng, Stankovic and Ramamritham 1986) and solved thanks to a bidding strategy.

So, in the next section of this paper, we will be concerned with a decision algorithm which, given any occurring task group $\mathcal{S}$, is capable of answering the question: "Can $\mathcal{S}$ be accepted?" The answer then depends on timing parameters of all the tasks running on the intended machine such as computation times and deadlines, and on the nature of precedence constraints among sporadic tasks. In addition to timing and precedence constraints, the scheduling strategy that is, the strategy of assigning the processor to tasks also affects the possibility of missing the deadlines and consequently the possibility of rejecting the new task group.

The remainder of the paper is organized as follows: In Section 2, we present the system model used in our study. Section 3 provides a background to real-time scheduling. In Section 4, we state new results about scheduling with precedence constraints. The Decision algorithm is described in Section 5. Conclusions are drawn in Section 6.

2. Task model

Before presenting our scheduling scheme, we would like to outline our model for real-time tasks. Indeed, the mandatory execution of any task within a critical time makes it necessary to model the timing characteristics of the tasks as well as the performance of the computing system in charge of their processing. An approach commonly used to verify feasibility (that is, absence of deadline missing) consists in creating an analytical model built around a fictitious set of tasks and hardware configuration.

To the monoprocessor machine is associated a periodic task set $\mathfrak{I}$ that runs over an infinite time. $\mathfrak{I}$ is described as follows: $\mathfrak{I} = \{T_i(s_i, C_i, R_i, P_i), i = 1 \text{ to } n\}$. In this characterization, task T_i initially makes a request for execution at time s_i, and so on at times $s_i + kP_i$ ($k = 1, 2, \ldots$) with P_i being the period of T_i. The execution time required for each request of T_i is C_i time units and a deadline occurs R_i time after each request by which T_i must have completed its execution.

Throughout our discussion, we assume that a preemptive discipline is employed. Thus, a request of C_i units of execution time can be satisfied by one or more quanta which add up to C_i.

There is no inter-task communication during periodic task execution and tasks don't use any critical resources. Overheads due to preemption and implementation of the scheduling strategy are evaluated and included in execution times of tasks. Let

$$u_i = \frac{C_i}{P_i}$$

be the utilization factor of T_i and

$$U = \sum_{i=1}^{n} u_i.$$

The condition $U \leq 1$ is necessary for the existence of a feasible execution for every request. We suppose that $\mathfrak{I}$ verifies this condition and we say that $\mathfrak{I}$ is a schedulable task set.

Moreover, some groups of aperiodic time critical tasks said to be sporadic may arrive at any moment and require to be run just once. Each group is a set of tasks $\mathcal{S}$ with $\mathcal{S} = \{S_i(r_i, C_i, d_i), i = 1 \text{ to } m\}$ and its related precedence graph G. Each task S_i is specified by its release time r_i, its execution time C_i and its deadline d_i. We will say that task S_i is ready at time t (that is, may be processed at time t) if condition $r_i \leq t$ is verified. While timing parameters of periodic tasks are well known at the initialization time, the system is aware of a sporadic task group after its arrival.

We also use $\mathcal{S}$ to refer to the set of vertices in G. So, graph G induces a partial order "$<$" on $\mathcal{S}$, defined by $S_i < S_j$ if and only if G contains a directed path from the node for S_i to the node for S_j. Then, we say that S_i is a predecessor of S_j (and S_j is a successor of S_i). Besides, we write $S_i \rightarrow S_j$ to indicate that G contains an arc directed from S_i to S_j and, in this case we say that S_i is an immediate predecessor of S_j (and S_j is an immediate successor of S_i). A group has one or more beginning tasks and ending tasks. A beginning task is one with no predecessor task. An ending task is one with no successor task.

3. Background

In addition to the timing parameters of tasks, the scheduling strategy will influence the performance of the computing system. Here, the machine uses a preemptive scheduling strategy which means that a processor executing a task may be assigned to execute another task without completing the previous one. Non-preemptive scheduling is certainly more practical because of its simplicity. Nevertheless, it is a NP-hard problem even for independent tasks (Garey and Johnson 1977) (for more details about the NP-Completeness theory see (Garey and Johnson 1979). A valid schedule must be constructed by suboptimal heuristics which lead to lower processor utilization and consequently to the rejection of more sporadic tasks than is case with preemptive scheduling. Priority scheduling is the dominant scheduling strategy in real-time systems. A static or dynamic priority is assigned to each task and whenever a scheduling decision is made, the task with the highest priority is selected.

3.1. Scheduling independent tasks

Scheduling time critical tasks with no precedence constraints has received much attention (see (Blazewicz, Cellary, Slowinsky and Weglarz 1986) for a survey). In case of a single processor, both Earliest Deadline (ED) (Liu and Layland 1973) and Least Laxity (Sorenson 1974) (scheduling the task with the least difference between its deadline and the expected completion time first) are optimal. Here, optimality means that they produce a valid schedule (that is, they feasibly schedule) for any schedulable time critical task set. Although Earliest Deadline and Least Laxity are both optimal in terms of scheduling performance, it must be noted that Earliest Deadline is more effective than Least Laxity in terms of numbers of preemptions and consequently overheads which are involved in their implementation.

Let $\mathcal{S} = \{S_i(r_i, C_i, d_i), i = 1 \text{ to } m\}$ be an aperiodic task set where each task is specified by its release time, execution time and deadline. The following lemma was stated in (Chetto and Chetto 1987):

LEMMA 1. $\mathcal{S}$ is feasibly scheduled by ED if and only if $\forall j = 1 \ldots m$, $\forall i = 1 \ldots m$ such that $r_i \leq r_j$, $d_i \leq d_j$,

$$\sum_{\substack{r_k \leq r_i \\ d_k \leq d_j}} C_k \leq d_j - r_i \tag{1}$$

Let $\mathfrak{I} = \{T_i(s_i, C_i, R_i, P_i), i = 1 \text{ to n}\}$ be a periodic task set defined as previously. In (Liu and Layland 1973) and (Labetoulle 1974), the result of lemma 2 was proved:

LEMMA 2. $\mathfrak{I}$ is feasibly scheduled by ED if

$$\sum_{i=1}^{n} \frac{C_i}{R_i} \leq 1 \tag{2}$$

Verification of feasibility for $\mathfrak{I}$ can be performed by testing inequality (2) which provides only a sufficient condition. A necessary and sufficient condition can be obtained by constructing a schedule using the ED strategy and checking to see if the schedule so constructed is valid. For a synchronous task set (that is, $s_i = s_j$ for all $1 \leq i, j \leq n$), it suffices to produce the schedule within $[0, P]$ where P is the least common multiple of $\{P_1, P_2, \ldots P_n\}$. However, for an asynchronous task set (that is, $s_i \neq s_j$ for some $1 \leq i, j \leq n$), the schedule must be constructed from 0 to $s + 2P$ where $s = \max \{s_1, s_2, \ldots s_n\}$ (Leung and Merril 1980). The time bound $s + 2P$ comes from the property of cyclicity that characterizes the schedule produced by any preemptive strategy for any periodic task set. This means that the processor does exactly the same thing at time $t (t \geq s + P)$ that it does at time $t + kP$ $(k = 2, 3 \ldots)$. So, studying the form of the schedule produced over an infinite time length amounts to studying the form of the schedule produced over the time interval $[s + P, s + 2P]$ and to replicating this schedule on the subsequent windows $[s + kP, s + (k + 1)P]$, $k = 2, 3 \ldots$.

To sum up, if a machine only supports periodic tasks, it becomes easy to know exactly what is done by the processor at any moment since the schedule which is to be executed can be determined at system initialization time and is not modified during the lifetime of the system.

However, if we assume that aperiodic tasks (said to be sporadic) may crop up dynamically at certain points in time, it becomes imperative to check in line whether there can exist a valid schedule since any new occurring task can affect the scheduling of periodic tasks and previously accepted sporadic tasks and provoke a deadline missing. A solution for this dynamic decision problem was proposed recently in (Chetto and Chetto 1989). Periodic and sporadic tasks are scheduled according to a common optimal strategy such as ED. Let τ be the current time which coincides with the occurring time of a sporadic task. Let D be the latest deadline of sporadic tasks supported by the machine at time τ, and d be the deadline of the occurring sporadic task. It was proved that the question of the acceptance of this task is equivalent to the question of the existence of a valid schedule within $[d, D + P]$. To answer this question, it suffices to gather all the requests of periodic tasks and sporadic tasks available within $[\tau, D + P]$ and thus, verifying that all the deadlines greater than or equal to d can be met. Determination of a necessary and sufficient condition for the acceptance of a new task then directly comes from the feasibility test of a set of aperiodic tasks given by relation (1). Then, each task S_k represents either a request of periodic task or a sporadic task and C_k is the remaining execution time of S_k at the current time τ. In summary, when sporadic tasks are assumed to be independent, we can test the acceptance of a new task by performing an algorithm which runs in $O(N^2)$ where N

is bounded by the total number of sporadic tasks and requests of periodic tasks available in the window $[\tau, D + P]$. This complexity can be reduced to $O(N)$ if any sporadic task is ready to be processed as soon as it arrives at the machine.

3.2. Scheduling dependent tasks

The problem of scheduling dependent aperiodic tasks on a single processor machine may be stated as follows: there is a given task set $\mathcal{S}$ as described previously and a partial order $<$ on $\mathcal{S}$, and the objective is to find a valid schedule for $\mathcal{S}$ (that is, a schedule which meets both timing and precedence constraints). In this context, a task set is said to be schedulable if there exists at least one valid schedule for $\mathcal{S}$.

When the tasks in $\mathcal{S}$ are not preemptable and are ready at time $t = 0$ (that is, for all $1 \leq i \leq m$, $r_i = 0$), an optimal scheduling algorithm exists and is given in (Lawler and Moors 1969). However, if tasks become ready at different time instants, a valid schedule cannot be found in polynomial time since the problem is NP-hard even without precedence constraints (Lenstra, Rinnooykan and Brucker 1977). When tasks are preemptable, an optimal scheduling algorithm exists for tasks which become ready at different time instants (that is, $r_i \neq r_j$ for some $1 \leq i, j \leq m$). This algorithm was presented in (Blazewicz 1976) and runs as follows: a modified deadline d_i^* is attributed to every task S_i according to the following formula:

$$d_i^* = min\ \{d_i, \min\ (d_j;\ S_i < S_j)\}. \tag{3}$$

At any moment, the processor is assigned to the ready task S_i which has the minimum value of d_i^* and predecessors of S_i have been processed before or during time instant t. The complexity of this scheduling algorithm is $O(m^2)$. It so happens that the feasibility test for a set of dependent aperiodic tasks can be implemented in at most $O(m^2)$ time since it amounts to construct the schedule and check to see if all the deadlines are met. Clearly, such an implementation will require the use of two lists of tasks: list $\mathcal{L}_r$ of ready tasks (at any time t, $S_i \in \mathcal{L}_r$ if $r_i \leq t$) and list $\mathcal{L}_a$ of available tasks (at any time t, $S_i \in \mathcal{L}_a$ if $S_i \in \mathcal{L}_r$ and all the predecessors of S_i have completed execution at time t). List $\mathcal{L}_a$ gathers all the tasks waiting for the processor to be processed and is ordered according to the modified deadlines. In this environment, timing information on every task such as its release time and deadline is maintained in a data structure which is used by the dispatcher to select the task for execution. Precedence relations among tasks may be specified in one of the two following ways:

- Synchronization operations are explicitly implemented in the code program of every task which makes transparent the precedence relations from the point of view of the scheduler.
- For every task, a number of variables are used by the scheduler, each of them associated with one of its predecessor, and whenever a predecessor completes execution, the variable is updated and the task may become available if all its predecessors have been completed.

4. New results about dependent task scheduling

In the next section, we will deal with the problem of scheduling dependent tasks together with independent tasks. Therefore, for practical purposes, we have deemed preferable to propose a method that transforms a set of dependent tasks into a set of independent tasks so that a valid schedule can be obtained for the former if and only if a valid schedule can be obtained for the latter. This work method presents the following advantages:

- A unique scheduler employing the Earliest Deadline strategy may be used to schedule dependent and independent tasks without discrimination.
- The precedence constraints among tasks are taken into account by an adequate modification of timing parameters of every task. We will show that in any schedule produced by the Earliest Deadline algorithm, this modification ensures that the precedence constraints are obeyed.
- Since a set of dependent tasks can be viewed as a set of independent tasks with modified parameters, we may call upon relation (1) to test the existence of a valid schedule. This is an important aspect of the method since it is no longer necessary to construct the schedule in order to prove that all the deadlines can be met.

So, the algorithm we present consists of two basic parts. In the first part, modified parameters are computed and, in the second part, they are used to construct a priority list. This list is special in that both timing and precedence constraints are obeyed for any schedulable task set. We now will give a detailed description of this scheduling algorithm and then state a fundamental theorem.

The two basic ideas of the algorithm are that:

- the relative urgency of a task depends both on its deadline and on the deadlines of its successors;
- the starting time of a task depends both on its release time and on the completion time of its predecessors.

So, we shall combine these factors in a simple way to form, for each task, a modified deadline and a modified release time.

Let S_i and S_j be two aperiodic tasks in $\mathcal{S}$ such that $S_i \rightarrow S_j$. In any valid schedule that meets all the deadlines, the following relations must be satisfied: $f_i \leq d_i$ and $f_i \leq d_j - C_j$ where it is denoted by f_i the completion time of S_i. Clearly, not only must S_i be completed by time d_i, it must also be completed by time $d_j - C_j$ which represents the latest start time of its immediate successor S_j. Consequently, we can replace the deadline d_i of S_i by $\min(d_i, d_j - C_j)$ without changing the problem. The algorithm which constructs the modified deadlines can be implemented easily to run in at most $O(m^2)$ time for a general precedence graph:

1. For every ending task of the partial order, set $d_i^* := d_i$
2. *Select a task* S_i which has not been assigned its modified deadline d_i^* and whose immediate successors have been assigned their modified deadline. If no such task exists, halt.

3. Set $d_i^* := \min (d_i, \min (d_j^* - C_j ; S_i \rightarrow S_j))$
and return to step 2. (4)

Now, let us describe how to modify the release times. If $S_j \rightarrow S_i$, then for any valid schedule that meets the precedence constraints, the following conditions must be satisfied: $k_i \geq k_j + C_j$ and $k_i \geq r_i$ where it is denoted by k_i the starting time of S_i. Therefore, we can replace the release time r_i of S_i by $\max (r_i, r_j + C_j)$ without changing the problem. The algorithm which modifies the release times can be implemented in $O(m^2)$ time and runs as follows:

1. For every beginning task of the partial order, set $r_i^* := r_i$.
2. Select a task S_i which has not been assigned its modified release time r_i^* and whose immediate predecessors S_j have been assigned their modified release time. If no such task exists, halt.
3. Set $r_i^* := \max (r_i, \max (r_j^* + C_j; S_j \rightarrow S_i))$
and return to step 2. (5)

Finally, we obain an equivalent scheduling problem with modified release times $\{r_i^*\}$ and modified deadlines $\{d_i^*\}$ for which $r_i^* < r_j^*$ and $d_i^* < d_j^*$ whenever $S_i < S_j$ (Verification is left to the reader).

The second part of the algorithm then constructs a valid schedule, if such a schedule exists. It consists in applying the Earliest Deadline strategy to a set of tasks considered to be independent and supplied with new timing parameters which are their modified release times and deadlines. This means that a priority list $\mathcal{L}_r$ is formed by sorting the tasks according to their modified deadlines, so that, at any time t, $S_i \in \mathcal{L}_r$ if $r_i^* \geq t$. Now, let us prove that the algorithm is optimal (that is, produces a schedule which meets both precedence and timing constraints, if such a schedule exists).

THEOREM 1.[1] Let $\mathcal{S} = \{S_i(r_i, C_i, d_i), i = 1 \text{ to } m\}$ and $<$ be a partial order on $\mathcal{S}$. Let $\mathcal{S}^* = \{S_i^* (r_i^*, C_i^*, d_i^*), i = 1 \text{ to } m\}$ be a set of independent tasks such that, for all $1 \leq i \leq m$, $C_i^* = C_i$, d_i^* and r_i^* are given by formulas (4) and (5).

$\mathcal{S}$ is a schedulable if and only if $\mathcal{S}^*$ is schedulable.

Proof. (Only if part): Suppose $\mathcal{S}$ is schedulable. So, a schedule for $\mathcal{S}$ can be found which meets both precedence constraints and timing constraints. Let k_i be the starting time of S_i in this schedule. S_i starts execution after all its predecessors have been completed. Through the reasoning described above, the relationship $k_i \geq r_i^*$ is always verified where r_i^* is given by formula (5). Consequently, if S_i had initially been assigned a release time equal to r_i^*, the schedule would have been identical to the one obtained with r_i.

Let f_i be the completion time of S_i. Since the schedule is valid, S_i completes execution before the latest start time of its successors. Through the reasoning described previously, the relationship $f_i \leq d_i^*$ is always verified where d_i^* is given by formula (4). So, if S_i had initially been assigned a deadline equal to d_i^*, the schedule would have been identical to the one obtained with d_i.

In conclusion, we can state that each task S_i is processed within $[r_i^*, d_i^*]$. The schedulability of the ordered set $\mathcal{S}$ implies the schedulability of the associated set $\mathcal{S}^*$ of independent tasks.

(If part): Suppose $\mathcal{S}^*$ is schedulable. This means that at least one valid schedule can be found where each task S_i^* starts at or after time r_i^* and is completed at or before time d_i^*. As $r_i^* \geq r_i$ and $d_i^* \leq d_i$ the schedulability of the set of independent tasks $\mathcal{S}^*$ implies the adherence with the timing requirements of $\mathcal{S}$. We shall now prove there exists a valid schedule which verifies the precedence order on $\mathcal{S}$. Remember that for any $1 \leq i \leq m$ such that $S_i < S_j$, we have $r_i^* < r_j^*$ and $d_i^* < d_j^*$. As ED is optimal for scheduling independent tasks, it may be used to construct a schedule for $\mathcal{S}^*$. Then, at any moment, the task that is to be processed has the earliest deadline d_i^* among the ready tasks. A combination of this property and the two previous relations ensure that the start time of a task is anterior to the start time of any of its successors and no task can be preempted by one of its successors. In conclusion, precedence relations are always obeyed.

As both timing and precedence constraints of $\mathcal{S}$ can be met by scheduling $\mathcal{S}^*$ according to ED, this means that there exists at least one valid schedule for $\mathcal{S}$ and, consequently, $\mathcal{S}$ is schedulable.

COROLLARY 1. Let $\mathcal{S} = \{S_i(r_i, C_i, d_i), i = 1 \text{ to } m\}$ and $<$ be a partial order on $\mathcal{S}$. For all $1 \leq i \leq m$, let d_i^* and r_i^* be given by formulas (4) and (5).

$\mathcal{S}$ is schedulable if and only if $\forall j = 1 \ldots m$, $\forall i = 1 \ldots m$ such that $r_i^* \leq r_j^*$, $d_i^* \leq d_j^*$,

$$\sum_{\substack{r_k^* \leq r_i^* \\ d_k^* \leq d_j^*}} C_k \leq d_j^* - r_i^* \tag{6}$$

Proof. From theorem 1, $\mathcal{S}$ is schedulable if and only if $\mathcal{S}^*$ is schedulable (that is, feasibly scheduled according to ED). From lemma 1, we immediately deduce that relations (6) provide a feasibility test for $\mathcal{S}$.

5. The decision algorithm

Having described a particular approach to construct a schedule for a set of dependent aperiodic tasks and therefore having proposed a feasibility test for such a set, we are now prepared to return to our original goal, that of scheduling sporadic tasks (that is, groups of tasks that arrive dynamically). More precisely, we will be concerned with the set of preemptable periodic tasks $\mathcal{J}$ which are scheduled according to ED on one processor. The latter may accept to process additional tasks that arrive in groups at different instants, depending on its surplus processing power. Our objective here is to propose a test able to answer *yes* each time it is possible to the question of the acceptance of a newly occurring task group. Clearly, this test will consist in verifying whether there exists a schedule so that no task will be late and all the precedence constraints will be satisfied.

Let $\mathcal{S}^\tau$ be a sporadic task group arriving at time τ. Upon its arrival, $\mathcal{S}^\tau$ is characterized by a set of time critical tasks with precedence constraints. It may be described as follows: $\mathcal{S}^\tau = \{S_i^\tau(r_i, C_i, d_i), i = 1 \text{ to } m^\tau\}$. For each task S_i^τ, we have $r_i \geq \tau$ and $r_i + C_i \leq d_i$. At time τ, the local scheduler is described by the list of tasks waiting for the processor which have not completed their execution. Each task in this list is either the current request of a periodic task or a sporadic task previously accepted. It is characterized by a dynamic execution time, that is, a remaining execution time at τ, a deadline and a release time.

All the tasks, periodic and sporadic are scheduled according to the common ED strategy. This means that as soon as a task group arrives, modified timing parameters r^* and d^* are attributed to each task S in this group. Consequently, all the precedence constraints are made transparent and will be verified at any moment.

Thanks to this work method, we have transformed a problem of scheduling dependent tasks into a problem of scheduling independent tasks thus enabling us to derive a decision test from results previously established.

LEMMA 3. Let S be an occurring sporadic task with deadline d. Let D be the latest deadline of sporadic tasks (including task S). Let P be the least common multiple of the periods of periodic tasks. Task S is accepted if and only if there exists a valid schedule within $[d, D + P]$.

Proof. (See Chetto and Chetto 1989).

THEOREM 2. Let $\mathcal{S}^\tau$ be an occurring sporadic task group. Let $mind^*$ be the earliest modified deadline in this group. Let D^* be the latest modified deadline of sporadic tasks (including tasks of $\mathcal{S}^\tau$) and P be the least common multiple of the periods of periodic tasks. $\mathcal{S}^\tau$ is accepted if and only if there exists a valid schedule within $[mind^*, D^* + P]$.

Proof. $\mathcal{S}^\tau$ can be considered to be a set of independent tasks whose timing parameters have been modified according to formulas (4) and (5). $\mathcal{S}^\tau$ is accepted if and only if any task of $\mathcal{S}^\tau$ is accepted. The proof becomes immediate by applying lemma 3 to each task of $\mathcal{S}^\tau$.

Thanks to this theorem, the question of the acceptance of a new task group has been reduced to the question of the existence of a valid schedule. Now, to answer this question, let us examine the tasks which are available for processing in $[\tau, D^* + P]$.

(1) Since the sporadic tasks present on the machine at time τ are those which have not completed their execution, there are a certain number of them for which $r_i^* \leq \tau$ and possibly have been partially executed. So, each of them can be assimilated to an aperiodic task with a release time equal to τ and its execution time equal to the dynamic execution time of the sporadic task. The others are characterized by their release time r_i^* and their initial execution time C_i. In addition, any sporadic task is specified by a modified deadline.

(2) There are a certain number of requests of periodic tasks that are available within $[\tau, D^* + P]$. Let us consider solely those for whom the deadline is less than or equal to $D^* + P$. They correspond to pairs of values of j and k such that, either $s_j + kP_j \geq \tau$

and $r_j + R_j + kP_j \leq D^* + P$ or $r_j + kP_j < \tau$ and $r_j + R_j + kP_j > \tau$. The pair (j, k) refers to the k-th computation of the periodic task T_j. In the first case, a task can be associated to each request for which the execution has not already started at time τ. Consequently, the execution time for this task is equal to the execution time of the periodic task T_j. In the second case, a task can be associated to it for which the release time is equal to τ and the execution time is equal to the dynamic execution time of the periodic task T_j at time τ.

It follows that we can test for the existence of a valid schedule in $[mind^*, D^* + P]$ by formulating a scheduling problem for independent aperiodic tasks in $[\tau, D^* + P]$ and thus verifying that all the deadlines greater than or equal to $mind^*$ can be met.

Let us denote by $\mathcal{S}^*$ the task set which gathers all the sporadic tasks and requests of periodic tasks with a deadline less than or equal to $D^* + P$.

$\mathcal{S}^* = \{S_i^*(r_i^*, C_i^*, d_i^*), i = 1 \text{ to } N\}$. By assumption, $\forall i \in \{1 \ldots N\}$, $r_i^* \geq \tau$ and $d_i^* \leq D^* + P$. For convenience we will assume that $\mathcal{S}^*$ is an ordered set such that $i < j$ implies $d_i^* \leq d_j^*$ and the sporadic task with the earliest modified deadline in $\mathcal{S}^\tau$ has been inserted into $\mathcal{S}^*$ with index q. Now, we are concerned with the feasibility test for a set of independent aperiodic tasks on a monoprocessor machine and we may state the following result.

COROLLARY 2. The task group $\mathcal{S}^\tau$ is accepted if and only if $\forall j \in \{q \ldots N\}$, $\forall i \in \{1 \ldots j\}$ such that $r_i^* \leq r_j^*$

$$\sum_{\substack{k=1 \\ r_k^* \geq r_i^*}}^{j} C_k \leq d_j^* - r_i^* \tag{6}$$

Proof. From lemma 1 and theorem 2, the proof is immediate.

The above theorem enables us to perform the acceptance test of the task group $\mathcal{S}^\tau$ by an algorithm which runs in $O(N^2)$ where N denotes the total number of sporadic tasks and requests of periodic tasks available within $[\tau, D^* + P]$. Let m be the number of sporadic tasks at time τ. It follows that $N \leq m + p$ with

$$p = \left\lceil \frac{D^* + P - mind^*}{P} \right\rceil .$$

By $\lceil x \rceil$ the smallest integer greater than or equal to x is denoted.

When a sporadic task group occurs on the processor, a routine is invoked and is implemented by the following steps:

1. For any task in the group, the computation of its modified deadline and its modified release time.
2. The temporary introduction of the new tasks at the local scheduler.

3. Updating of all the dynamic properties of tasks and test of the acceptance conditions. If the new group cannot be accepted then all the tasks of this group are removed from the scheduler.

An outline of this routine can be described as follows:

```
begin
  -Let τ be the current clock time
  -Let P be the least common multiple of periods
  -Let 𝕊^τ be the occurring task group and G be the precedence graph associated to 𝕊^τ

  step 1 (* modification of timing parameters*)
        For each task S_i^τ in 𝕊^τ, from graph G, compute its modified release time r_i^* and its modified deadline d_i^*

  step 2 (* initialization of data structures*)
        -Let mind* be the least modified deadline of sporadic tasks in 𝕊^τ
        -Let D* be the greatest modified deadline of all the sporadic tasks (including the new occurring ones)
        -Let ℒ* be a transient list initially empty
        -Form 𝕊* as the list of all the sporadic tasks and requests of periodic tasks with deadline less than
        or equal to D* + P, ordered by non-decreasing deadlines.
        -Let q be the entry in 𝕊* such that mind* = d_q^* and be the current entry

  step 3 (* test of the acceptance conditions*)
        For each entry j in 𝕊* from the current entry to the last entry
           do
              -find the entry k in ℒ* such that r_j^* < r_k^*
              -consider S_j^* as inserted prior to the entry k, and be the current entry in ℒ*
              -N(k,j):=0
              For each entry i in ℒ* from the current entry down to the first entry
                 do
                       N(i,j):=N(i+1,j) + C_i
                       if N(i,j)>d_j^*−r_i^* then return (* the task group is not accepted*) end if
                 end do
           end do
        return (* the task group is accepted*)
end.
```

6. Conclusion

In the first part of this paper, we considered the problem of scheduling dependent and preemptable tasks with different release times, on one processor in a hard real-time environment. In this environment, all the tasks have strict deadlines that must be respected. We proposed to solve this problem by using an approach which consists in modifying timing parameters of tasks and then in applying the well known Earliest Deadline strategy. We proved that such an algorithm is optimal (that is, constructs a schedule which meets both timing and precedence constraints whenever it is possible).

The final goal of this paper was to propose a solution to a scheduling problem specific to dynamic hard real-time systems. It is assumed that a monoprocessor machine must run periodic tasks for the entire lifetime of the application and in addition, as many sporadic

tasks as possible which arrive in the processor in groups. Their occurrence date is assumed to be unknown and completely unpredictable at system initialization time. Furthermore, the machine is required to execute them either completely or not at all. From the previous results, we have proposed a test that permits us to determine on line whether or not an occurring task group can be accepted. The algorithm that implements such a test runs in polynomial time and is optimal since any task group which can be feasibly scheduled on the machine is accepted. The most important aspects of this approach is that it can be employed by any operating system whose scheduler uses the famous Earliest Deadline strategy.

Note

1. One of the referees has signaled us that an analogous result was obtained by Henn ([1978], Satz 3).

Acknowledgment

The authors wish to thank Dr. German and Dr. Perrillon for their suggestions that improved the readability of the paper.

References

Blazewicz, J. 1976. Scheduling dependent tasks with different arrival times to meet deadlines. In E. Gelenbe, H. Beilner (eds), *Modelling and Performance Evaluation of Computer Systems*, Amsterdam, North-Holland, 1976 pp 57–65.

Blazewicz, J., Cellary, W., Slowinsky, R., and Weglarz, J. 1986. Scheduling under resource constraints — Deterministic models. In P.L. Hammer (eds), *Annals of Operations Research*, Vol 7. Basel, J.C. Baltzer AG.

Cheng, S., Stankovic, J.A., and Ramamritham, K. 1986. Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real time systems. In *Proc. Real-time Systems Symposium*, (Feb.), Louisiana, pp 285–298.

Chetto, H., and Chetto, M. 1987. How to insure feasibility in distributed system for real-time control. *International Symposium on High Performance Computer Systems*, Paris, (Dec.).

Chetto, H., and Chetto, M. 1989. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15, (10) (Oct.): 1261, 1269.

Chetto, H. and Chetto, M. 1989. Scheduling periodic and sporadic tasks in a real time system. *Info. Proc. Letters*, 30, (4):177–184.

Garey, M.R. and Johnson, D.S. 1977. Two-processor scheduling with start times and deadlines. *SIAM J. on Computing*, 6:416–426.

Garey, M.R. and Johnson, D.S. 1979. *Computer and Intractability: A Guide to the Theory of NP-completeness.* San Francisco: Freeman.

Henn, R. 1978. Antwortzeitgesteuerte Prozessorzuteilung unter Strengen Zeitbedingungen. 1978. *Computing* (19):202–220.

Labetoulle, J. 1974. Some theorems on real-time scheduling. In E. Gelenbe and R. Mahl (eds) *Computer Architecture and Networks*, Amsterdam, North-Holland, pp 285–298.

Lawler, E.L. and Moors, J.M. 1969. A functional equation and its application to resource allocation and scheduling problems. *Management science*, 16, (1):77–84.

Lenstra, J.K., Rinnooykan, A.H.G., and Brucker, P. 1977. Complexity of machine scheduling problems. *Ann. Discrete Math.* 1:343–362.

Leung, J.Y.T., and Merril, M.L. 1980. A note on preemptive scheduling of periodic real-time tasks. *Info. Proc. Letters*, 11, (3): 115–118.

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

Ramamritham, K., and Stankovic, J.A. 1984. Dynamic task scheduling in distributed hard real-time systems. *IEEE Software*, (July):65–75.

Serlin, O. 1972. Scheduling of time critical processes. In *Proceedings of the Spring Joint Computers Conference*, 40:925–932.

Sorenson, P.G. 1974. A methodology for real-time system development. P.H.D. University of Toronto, Canada 1974.

Stankovic, J.A. 1988. A serious problem for next-generation systems. *IEEE Computer* 21, (10):10–19.

Zhao, W. and Ramamritham, K. 1985. Distributed scheduling using bidding and focussed addressing. In *Proc. Real-Time Syst. Symposium*, San Diego, (Dec. 3–6):103–111.