Optimal Reward-Based Scheduling for Periodic Real-Time Tasks

Hakan Aydin, *Student Member*, *IEEE*, Rami Melhem, *Fellow*, *IEEE*, Daniel Mossé, *Member*, *IEEE*, and Pedro Mejía-Alvarez

Abstract—Reward-based scheduling refers to the problem in which there is a reward associated with the execution of a task. In our framework, each real-time task comprises a mandatory and an optional part. The mandatory part must complete before the task's deadline, while a nondecreasing reward function is associated with the execution of the optional part, which can be interrupted at any time. Imprecise computation and Increased-Reward-with-Increased-Service models fall within the scope of this framework. In this paper, we address the reward-based scheduling problem for periodic tasks. An optimal schedule is one where mandatory parts complete in a timely manner and the weighted average reward is maximized. For linear and concave reward functions, which are most common, we 1) show the existence of an optimal schedule where the optional service time of a task is constant at every instance and 2) show how to efficiently compute this service time. We also prove the optimality of Rate Monotonic Scheduling (with harmonic periods), Earliest Deadline First, and Least Laxity First policies for the case of uniprocessors when used with the optimal service times we computed. Moreover, we extend our result by showing that any policy which can fully utilize *all* the processors is also optimal for the multiprocessor periodic reward-based scheduling. To show that our optimal solution is pushing the limits of reward-based scheduling, we further prove that, when the reward functions are convex, the problem becomes NP-Hard. Our static optimal solution, besides providing considerable reward improvements over the previous suboptimal strategies, also has a major practical benefit: Run-time overhead is eliminated and existing scheduling disciplines may be used without modification with the computed optimal service times.

Index Terms—Real-time systems, imprecise computation, periodic task scheduling, deadline scheduling, reward maximization.

1 Introduction

In a real-time system, each task must complete and produce correct output by the specified deadline. However, if the system is overloaded, it is not possible to meet each deadline. In the past, several techniques have been introduced by the research community regarding the appropriate strategy to use in overloaded systems of periodic real-time tasks.

One class of approaches focuses on providing somewhat less stringent guarantees for temporal constraints. In [16], some instances of a task are allowed to be skipped entirely. The *skip factor* determines how often instances of a given task may be left unexecuted. A best effort strategy is introduced in [11], aiming at meeting k deadlines out of n instances of a given task. This framework is also known as *(n, k)-firm deadlines* scheme. Bernat and Burns present in [2] a hybrid and improved approach to provide hard real-time guarantees to k out of n consecutive instances of a task.

The techniques mentioned above tacitly assume that a task's output is of no value if it is not executed completely. However, in many application areas such as multimedia applications [25], image and speech processing [5], [6], [9], [27], time-dependent planning [4], robot control/navigation systems [12], [29], medical decision making [13], information gathering [10], real-time heuristic search [17], and database query processing [28], a partial or approximate but timely result is usually acceptable.

The *imprecise computation* [7], [19], [21] and *IRIS (Increased Reward with Increased Service)* [14], [15], [18] models were proposed to enhance the resource utilization and graceful degradation of real-time systems when compared with hard real-time environments where worst-case guarantees must be provided. In these models, every real-time task is composed of a mandatory part and an optional part. The former should be completed by the task's deadline to provide output of acceptable (minimal) quality. The optional part is to be executed after the mandatory part, while still before the deadline, if there are enough resources in the system that are not committed to running mandatory parts for any task. The longer the optional part executes, the better the quality of the result (the higher the reward).

The algorithms proposed for imprecise computation applications concentrate on a model that has an upper bound on the execution time that could be assigned to the optional part [7], [21], [26]. The aim is usually to minimize the (weighted) sum of errors. Several efficient algorithms have been proposed to solve optimally the scheduling problem of aperiodic imprecise computation tasks [21], [26]. A common assumption in these studies is that the quality of the results produced is a *linear* function of the precision error; consequently, the possibility of having more general error functions is usually not addressed.

An alternative model allows tasks to get increasing reward with increasing service (IRIS model) [14], [15], [18]

- *H. Aydin, R. Melhem, and D. Mossé are with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: {aydin, melhem,mosse}@cs.pitt.edu.*
- *P. Mejía-Alvarez is with CINVESTAV-IPN. Sección de Computación, Av. I.P.N. 2508, Zacatenco, México, DF. 07300. E-mail: pmejia@computacion.cs.cinvestav.mx.*

Manuscript received 25 Feb. 2000; accepted 23 Oct. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111563.

without an upper bound on the execution times of the tasks (although the deadline of the task is an implicit upper bound) and without the separation between mandatory and optional parts [14]. A task executes for as long as the scheduler allows it to, before its deadline. Typically, a nondecreasing *concave* reward function is associated with each task's execution time. In [14], [15], the problem of maximizing the total reward in a system of aperiodic independent tasks is addressed. The optimal solution with static task sets is presented, as well as two extensions that include mandatory parts and policies for dynamic task arrivals.

Note that the imprecise computation and IRIS models are closely related since the performance metrics can be defined as duals (maximizing the total reward vs. minimizing the total error). Similarly, a concave reward function corresponds to a convex error function and vice versa. We use the term "Reward-based scheduling" to encompass scheduling frameworks, including Imprecise Computation and IRIS models, where each task can be logically decomposed into a mandatory and an optional subtask. A nondecreasing reward function is associated with the execution of each optional part.

An interesting question concerns the types of reward functions that represent realistic application areas. A *linear* reward function [19], [21] models the case where the benefit to the overall system increases *uniformly* during the optional execution. Similarly, a *concave* reward function [14], [15], [18], [25] addresses the case where the greatest increase in reward (i.e., refinement in the output quality) is obtained during the first portions of optional executions. Linear and general concave functions are considered as the most realistic and typical in the literature since they adequately capture the behavior of many application areas like image and speech processing [5], [6], [9], [27], multimedia applications [25], time-dependent planning [9], robot control/navigation systems [29], real-time heuristic search [17], information gathering [10], and database query processing [28]. For completeness, in this paper, we show that the case of *convex* reward functions is an NP-Hard problem and thus focus on linear and concave reward functions. Reward functions with 0/1 constraints, where no reward is accrued unless the *entire* optional part is executed, or as step functions have also received some interest in the literature. Unfortunately, this problem has been shown to be NP-Complete in [26].

Periodic reward-based scheduling remains relatively unexplored, since the important work of Chung et al. [7]. In that paper, the authors classified the possible application areas as "error noncumulative" and "error cumulative." In the former, errors (or optional parts left unexecuted) have no effect on the future instances of the same task. Well-known examples of this category are tasks which receive, process and transmit periodically audio, video or compressed images [5], [6], [9], [25], [27] and information retrieval tasks [10], [28]. For instance, a real-time video application where, at each period, an imperfect but decent quality image is produced first from received data and subsequently refined within the capacity of the available resources, can be readily modeled by an error noncumulative reward-based task. In "error cumulative" applications, such as radar tracking, an optional instance must be executed completely at every (predetermined) k invocations. The authors further proved that the case of error-cumulative jobs is an NP-Complete problem. In this paper, we restrict ourselves to error noncumulative applications.

Recently, a QoS-based resource allocation model (QRAM) has been proposed for periodic applications [25]. In that study, the problem is to optimally allocate several resources to the various applications such that they simultaneously meet their minimum requirements along multiple QoS dimensions and the total system utility is maximized. In one aspect, this can be viewed as a generalization of optimal CPU allocation problem to multiple resources and quality dimensions. Further, dependent and independent quality dimensions are separately addressed for the first time in that work. However, a fundamental assumption of that model is that the reward functions and resource allocations are in terms of *utilization of resources*. Our model follows the Imprecise Computation model more closely, where the reward accrued has to be computed separately over all task instances and the problem is to find the optimal service times for *each* instance and the optimal schedule with these assignments.

1.1 Aspects of the Periodic Reward-Based Scheduling Problem

The difficulty of finding an optimal schedule for a periodic reward-based task set has its origin in two objectives that must be simultaneously achieved, namely:

1. Meeting deadlines of mandatory parts at *every* periodic task invocation.
2. Scheduling optional parts to maximize the total (or average) reward.

These two objectives are both important, yet often incompatible. In other words, hard deadlines of mandatory parts may require sacrificing optional parts with great value to the system.

The analytical treatment of the problem is complicated by the fact that, in an optimal schedule, optional service times of a given task may *vary* from instance to instance, which makes the framework of classical periodic scheduling theory inapplicable. Furthermore, this fact introduces a large number of variables in any analytical approach. Finally, by allowing nonlinear reward functions to better characterize the optional tasks' contribution to the overall system, the optimization problem becomes computationally harder.

In [7], Chung et al. proposed the strategy of assigning *statically* higher priorities to mandatory parts. This decision, as proven in that paper, effectively achieves the first objective mentioned above by securing mandatory parts from the potential interference of optional parts. Optional parts are scheduled whenever no mandatory part is ready in the system. In [7], the simulation results regarding the performance of several policies which assign static or dynamic priorities among optional parts are reported. We call the class of algorithms that statically assign higher priorities to mandatory parts *Mandatory-First Algorithms*.

In our solution, we do *not* decouple the objectives of meeting the deadlines of mandatory parts and maximizing the total (or average) reward. We formulate the periodic reward-based scheduling problem as an optimization problem and derive an important and surprising property of the solution for the most common (i.e., linear and concave) reward functions. Namely, *we prove that there is always an optimal schedule where optional service times of a given task do not vary from instance to instance*. This important result immediately implies that the optimality (in terms of achievable utilization) of any policy which can fully use the processor in the case of hard-real time periodic tasks also holds in the context of reward-based scheduling (in terms of total reward) *when used with these optimal service times*. Examples of such policies are RMS-h (Rate Monotonic Scheduling with harmonic periods) [20], EDF (Earliest Deadline First) [20], and LLF (Least Laxity First) [23]. We also extend the framework to homogeneous multiprocessor settings and prove that any policy which can fully utilize all the processors is also optimal for scheduling periodic reward-based tasks (in terms of total reward) on multiprocessor environments.

Following these existence proofs, we address the problem of efficiently computing optimal service times and provide polynomial-time algorithms for linear and/or general concave reward functions. Note that using these optimal and constant optimal service times has also important practical advantages: 1) The runtime overhead due to the existence of mandatory/optional dichotomy and reward functions is removed and 2) existing RMS (with harmonic periods), EDF, and LLF schedulers may be used without any modification with these optimal assignments.

The remainder of this paper is organized as follows: In Section 2, the system model and basic definitions are given. The main result about the optimality of any periodic policy which can fully utilize the processor(s) is obtained in Section 3. In Section 4, we first analyze the worst-case performance of Mandatory-First approaches. We also provide the results of simulations run on a synthetic task set, in order to compare the performance of policies proposed in [7] against our optimal algorithm. Then, we examine whether the optimality of identical service times still holds if the model is modified by dropping some fundamental assumptions (Section 5). In Section 6, we show that the concavity assumption is also necessary for computational efficiency by proving that allowing convex reward functions results in an NP-Hard problem. We present details about the specific optimization problem that we use in Section 7. We conclude by summarizing our contribution and discussing future work.

2 System Model

We first develop and present our solution for uniprocessor systems, then we show how to extend it to the case of homogeneous multiprocessor systems.

We consider a set $\mathbf{T}$ of n periodic real-time tasks $T_1, T_2, \ldots, T_n$. The period of T_i is denoted by P_i, which is also equal to the deadline of the current invocation. We refer to the jth invocation of task T_i as T_{ij}. All tasks are assumed to be independent and ready at $t = 0$.

Each task T_i consists of a mandatory part M_i and an optional part O_i. The length of the mandatory part is denoted by m_i; each task must receive at least m_i units of service time before its deadline in order to provide output of acceptable quality. The optional part O_i becomes ready for execution only when the mandatory part M_i completes; it can execute as long as the scheduler allows before the deadline.

Associated with each optional part of a task is a reward function $R_i(t_{ij})$ which indicates the reward accrued by task T_{ij} when it receives t_{ij} units of service *beyond its mandatory portion*. $R_i(t_{ij})$ is of the form:

$$R_i(t_{ij}) = \begin{cases} f_i(t_{ij}) & \text{if} \quad 0 \leq t_{ij} \leq o_i \\ f_i(o_i) & \text{if} \quad t_{ij} > o_i, \end{cases} \tag{1}$$

where f_i is a nondecreasing, concave, and continuously differentiable function over nonnegative real numbers and o_i is the length of the *entire* optional part O_i. We underline that $f_i(t_{ij})$ is nondecreasing: The benefit of task T_{ij} cannot decrease by allowing it to run longer. Notice that the reward function $R_i(t)$ is not necessarily differentiable at $t = o_i$. Note also that, in this formulation, by the time the task's optional execution time t reaches the threshold value o_i, the reward accrued ceases to increase. Clearly, the reward of executing an optional part O_i for an amount of time $t_i > o_i$ will be the same as the reward for executing for o_i time units. Therefore, it is not beneficial to execute O_i for more than o_i time units since T_i has completed its mandatory and optional parts for that period.

Having nondecreasing concave reward functions means that, while a task T_i receives service beyond its mandatory portion M_i, its reward monotonically increases. However, its *rate of increase* decreases or remains constant with time. Note that the first derivative of a nondecreasing concave function is nonincreasing. The concavity assumption implies that the early portions of an optional execution are *not* less important than the later ones, which adequately captures many application areas mentioned in the introduction. We concentrate on concave reward functions, including linear reward functions.

A schedule of periodic tasks is **feasible** if mandatory parts meet their deadlines at every invocation. Given a feasible schedule of the task set **T**, the **average reward** of task T_i is defined as:

$$REW_i = \frac{P_i}{P} \sum_{j=1}^{P/P_i} R_i(t_{ij}), \tag{2}$$

where P is the *hyperperiod*, that is, the least common multiple of $P_1, P_2, \ldots, P_n$, and t_{ij} is the service time assigned to the jth instance of optional part of task T_i. That is, the average reward of T_i is computed over the number of its invocations during the hyperperiod P in an analogous way to the definition of average error in [7].[1]

The **average weighted reward** of a feasible schedule is then given by:

1. We note that the results we prove easily extend to the case in which one is interested in maximizing the *total* reward $\sum_{j=1}^{P/P_i} f_i(t_{ij})$.

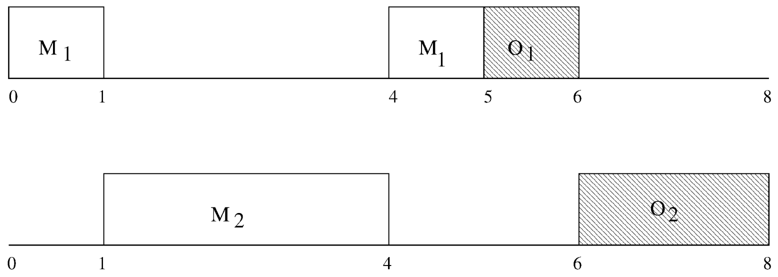


Fig. 1. A schedule produced by the Mandatory-First Algorithm.

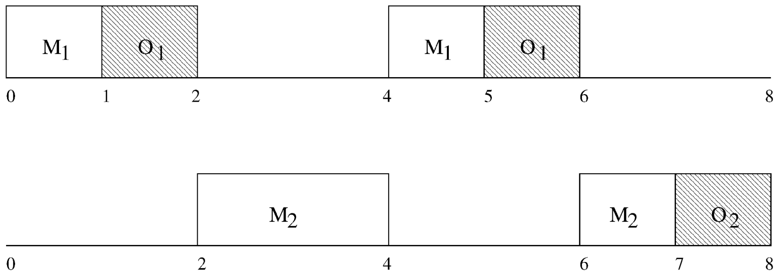


Fig. 2. An optimal schedule.

$$REW_W = \sum_{i=1}^{n} w_i \; REW_i, \qquad (3)$$

where w_i is a constant in the interval (0, 1], indicating the relative importance of optional part O_i. Although this is the most general formulation, it is easy to see that the weight w_i can always be incorporated into the reward function $R_i()$, by replacing it by $w_i\, R_i()$. Thus, we will assume that all weight (importance) information is already expressed in the reward function formulation and that REW_W is simply equal to $\sum_{i=1}^{n} REW_i$.

Finally, a schedule is **optimal** if it is feasible and it maximizes the average weighted reward.

2.1 A Motivating Example

Before describing our solution to the problem, we present a simple example based on a synthetic task set, which shows the performance limitations of *any* Mandatory-First algorithm. Consider two tasks where $P_1 = 4, m_1 = 1, o_1 = 1, P_2 = 8, m_2 = 3, o_2 = 5$. Assume that the reward functions associated with optional parts are linear and $f_1(t_1) = k_1 t_1, f_2(t_2) = k_2 t_2$. Furthermore, suppose that k_2 associated with the reward accrued by T_2 is negligible when compared to k_1 (i.e., $k_1 \gg k_2$). In this case, the "best" algorithm among "Mandatory-First" approaches should produce the schedule shown in Fig. 1.

Above, we assumed that the Rate Monotonic Priority Assignment is used whenever more than one mandatory tasks are simultaneously ready, as in [7]. Yet, following other (dynamic or static) priority schemes would not change the fact that the processor will be busy executing solely mandatory parts until $t = 5$ under any Mandatory-First approach. During the remaining idle interval [5, 8], the best algorithm would have chosen to schedule O_1 completely (which brings most benefit to the system) for one time unit and O_2 for two time units. However, an optimal algorithm would produce the schedule depicted in Fig. 2.

As can be seen, the optimal strategy in this case consisted of delaying the execution of M_2 in order to be able to execute "valuable" O_1 and we would still meet the deadlines of all mandatory parts. By doing so, we would succeed in executing two instances of O_1, in contrast to any Mandatory-First scheme which can execute only one instance of O_1. Remembering that $k_1 \gg k_2$, one can conclude that the reward accrued by the "best" Mandatory-First scheme may only be around half of that accrued by the optimal one, for this example. Also, observe that in the optimal schedule, the optional execution times of a given task did not vary from instance to instance. In the next section, we prove that this pattern is not a mere coincidence. We further perform an analytical worst-case analysis of Mandatory-First algorithms in Section 4.

3 Optimality of Full-Utilization Policies for Periodic Reward-Based Scheduling

We first formalize the Periodic Reward-Based Scheduling problem. The objective is finding optimal $\{t_{ij}\}$ values to maximize the average reward. By substituting the average reward expression given by (2) in (3), we obtain our objective function:

$$\text{maximize} \quad \sum_{i=1}^{n} \frac{P_i}{P} \sum_{j=1}^{P/P_i} R_i(t_{ij}).$$

The first constraint that we must enforce is that the total processor demand of mandatory and optional parts during the hyperperiod P may not exceed the available computing capacity, that is:

$$\sum_{i=1}^{n} \sum_{j=1}^{P/P_i} (m_i + t_{ij}) \leq P.$$

Note that this constraint is necessary, but by no means sufficient for feasibility of the task set with $\{m_i\}$ and $\{t_{ij}\}$

values. Next, we observe that optimal t_{ij} values may not be less than zero since negative service times do not have any physical interpretation. In addition, the service time of an optional instance of T_i does not need to exceed the upper bound o_i of reward function $R_i(t)$ since the reward accrued by T_i ceases to increase after $t_{ij} = o_i$. Hence, we obtain our second constraint set:

$$0 \leq t_{ij} \leq o_i \quad i = 1, \ldots, n \quad j = 1, \ldots, \frac{P}{P_i}.$$

The constraint above allows us to readily substitute $f_i()$ for $R_i()$ in the objective function. Finally, we need to express the "full" feasibility constraint, including the requirement that mandatory parts complete in a timely manner at every invocation. Note that it is sufficient to have one feasible schedule with the involved $\{m_i\}$ and optimal $\{t_{ij}\}$ values:

$$\text{There exists a feasible schedule with } \{m_i\} \text{ and } \{t_{ij}\} \text{ values.}$$

We express this constraint in English and not through formulas since the policy or algorithm producing this schedule including optimal t_{ij} assignments need not be specified at this point.

To recapture all the constraints, the periodic reward-based scheduling problem, which we denote by REW-PER, is to find $\{t_{ij}\}$ values so as to:

$$\text{maximize} \qquad \sum_{i=1}^{n} \frac{P_i}{P} \sum_{j=1}^{P/P_i} f_i(t_{ij}) \tag{4}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} \frac{P}{P_i} m_i + \sum_{i=1}^{n} \sum_{j=1}^{P/P_i} t_{ij} \leq P \tag{5}$$

$$0 \leq t_{ij} \leq o_i \quad i = 1, \ldots, n \quad j = 1, \ldots, \frac{P}{P_i} \tag{6}$$

$$\text{There exists a feasible schedule with } \{m_i\} \text{ and } \{t_{ij}\} \text{ values.} \tag{7}$$

Before stating our main result, we underline that if $\sum_{i=1}^{n} \frac{P}{P_i} m_i > P$, it is not possible to schedule mandatory parts in a timely manner and the optimization problem has no solution. Note that this condition is equivalent to $\sum_{i=1}^{n} \frac{m_i}{P_i} > 1$, which indicates that the task set would be unschedulable, even if it consisted of only mandatory parts. Hence, hereafter, we assume that $\sum_{i=1}^{n} \frac{m_i}{P_i} \leq 1$.

Theorem 1. *Given an instance of Problem REW-PER, there exists an optimal solution where the optional parts of a task T_i receive the* same *service time at every instance, that is, $t_{ij} = t_{ik}$ $1 \leq j < k \leq \frac{P}{P_i}$. Furthermore, any periodic hard-real time scheduling policy which can fully utilize the processor (EDF, LLF, RMS-h) can be used to obtain a feasible schedule with these assignments.*

Proof. Our strategy to prove the theorem will be as follows: We will drop the feasibility condition (7) and obtain a new optimization problem whose feasible region strictly contains that of REW-PER. Specifically, we consider a new optimization problem, denoted by MAX-REW, where the objective function is again given by (4), but only the constraint sets (5) and (6) have to be satisfied. Note that the new problem MAX-REW does *not* a priori correspond to any scheduling problem since the feasibility issue is not addressed. We then show that there exists an optimal solution of MAX-REW where $t_{ij} = t_{ik},\ 1 \leq j < k \leq \frac{P}{P_i}$. Then, we will return to REW-PER and demonstrate the existence of a feasible schedule (i.e., satisfiability of (7)) under these assignments. The reward associated with MAX-REW's optimal solution is always greater than or equal to that of REW-PER's optimal solution, for MAX-REW does *not* consider one of the REW-PER's constraints. This will imply that this specific optimal solution of the new problem MAX-REW is also an optimal solution of REW-PER.

Now, we show that there exists an optimal solution of MAX-REW where $t_{ij} = t_{ik},\ 1 \leq j < k \leq \frac{P}{P_i}$.

Claim 1. *Let $\{t_{ij}\}$ be an optimal solution to MAX-REW, $1 \leq i \leq n \quad 1 \leq j \leq \frac{P}{P_i} = q_i$. Then, $\{t'_{ij}\}$, where*

$$t'_{i1} = t'_{i2} = \ldots = t'_{iq_i} = t'_i = \frac{t_{i1} + t_{i2} + \ldots + t_{iq_i}}{q_i}$$
$$1 \leq i \leq n \quad 1 \leq j \leq q_i$$

is also an optimal solution to MAX-REW.

- We first show that $\{t'_{ij}\}$ values satisfy the constraints (5) and (6) if $\{t_{ij}\}$ already satisfy them. Since $\sum_{j=1}^{q_i} t_{ij} = \sum_{j=1}^{q_i} t'_{ij} = q_i t'_i$ the constraint (5) is not violated by the transformation. Also, by assumption, $t_{ij} \leq o_i\, \forall j$, which implies $\max_j\{t_{ij}\} \leq o_i$. Since t'_i, which is the arithmetic mean of $t_{i1}, t_{i2}, \ldots, t_{iq_i}$ is necessarily less than or equal to $\max_j\{t_{ij}\}$, the constraint set (6) is not violated either by the transformation.
- Furthermore, the total reward does not decrease by this transformation since $\sum_{j=1}^{q_i} f_i(t_{ij}) \leq q_i\ f_i(t'_i)$. The proof of this statement is presented in the Appendix.

Using Claim 1, we can commit to finding *an* optimal solution of MAX-REW by setting

$$t_{i1} = t_{i2} = \ldots = t_{iq_i} = t_i \quad i = 1, \ldots, n.$$

In this case, $\sum_{j=1}^{P/P_i} f_i(t_{ij}) = \frac{P}{P_i}\ f_i(t_i)$ and $\sum_{j=1}^{P/P_i} t_{ij} = \frac{P}{P_i}\ t_i$. Hence, this version of MAX-REW can be rewritten as:

$$\text{maximize} \qquad \sum_{i=1}^{n} f_i(t_i) \tag{8}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} \frac{P}{P_i} t_i \leq P - \sum_{i=1}^{n} \frac{P}{P_i} m_i \tag{9}$$

$$0 \leq t_i \leq o_i \quad i = 1, \ldots, n. \tag{10}$$

Finally, we prove that the optimal solution $t_1, t_2, \ldots, t_n$ of MAX-REW above automatically satisfies the feasibility constraint (7) of our original problem REW-PER. Having equal optional service times for a given task greatly simplifies the verification of this constraint. Since $t_1, t_2, \ldots, t_n$ (by assumption) satisfy (9), we can write $\sum_{i=1}^{n} P \cdot \frac{m_i+t_i}{P_i} \leq P$ or, equivalently, $\sum_{i=1}^{n} \frac{m_i+t_i}{P_i} \leq 1$.

This implies that any policy which can achieve 100 percent processor utilization in classical periodic scheduling theory (EDF, LLF, RMS-h) can be used to obtain a feasible schedule for tasks, which now have identical execution times $m_i + t_i$ at every instance. Hence, the "full feasibility" constraint (7) of REW-PER is satisfied. Furthermore, this schedule clearly maximizes the average reward since $\{t_i\}$ values maximize MAX-REW whose feasible region encompasses that of REW-PER. □

Corollary 1. *Optimal t_i values for the Problem REW-PER can be found by solving the optimization problem given by (8), (9), and (10).*

We present the details of the solution of this concave optimization problem in Section 7.

3.1 Extension to Multiprocessors

The existence proof of identical service times can be easily extended to homogeneous multiprocessors. The original formulation of REW-PER needs to be modified in order to reflect the multiprocessor environment. Note that the objective function (4), the lower and upper bound constraints (6) on optional service times, and the full feasibility constraint (7) can be kept as they are. However, with k processors, the system can potentially have a task set whose total utilization is k instead of 1. Hence, we need to change the first constraint accordingly.

By doing so, we obtain the formulation of periodic imprecise computation problem for k processors, denoted as MULTI-REW:

$$\text{maximize} \quad \sum_{i=1}^{n} \frac{P_i}{P} \sum_{j=1}^{P/P_i} f_i(t_{ij}) \tag{11}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{P}{P_i} m_i + \sum_{i=1}^{n} \sum_{j=1}^{P/P_i} t_{ij} \leq k \cdot P \tag{12}$$

$$0 \leq t_{ij} \leq o_i \quad i = 1, \ldots, n \quad j = 1, \ldots, \frac{P}{P_i} \tag{13}$$

There exists a feasible schedule on k processors with $\{m_i\}$ and $\{t_{ij}\}$ values. (14)

Following exactly the same line of reasoning depicted in Theorem 1, we can prove the following:

Theorem 2. *Given an instance of Problem MULTI-REW, there exists an optimal solution where the optional parts of a task T_i receive the* same *service time at every instance, that is, $t_{ij} = t_{ik}$ $1 \leq j < k \leq \frac{P}{P_i}$. Furthermore, any scheduling policy which can achieve full utilization on k processors can be used to obtain a feasible schedule with these assignments.*

An example of such full-utilization policies for multiprocessors is provided by Bertossi and Mancini in [3]. We note that the PFair scheduling policy [1], which can also achieve full-utilization, assumes that all the periods are multiples of the slot length and hence it can not be used in this context.

Corollary 2. *Optimal t_i values for the Problem MULTI-REW can be found by solving the following optimization problem:*

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(t_i) \tag{15}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{P}{P_i} t_i \leq k \cdot P - \sum_{i=1}^{n} \frac{P}{P_i} m_i \tag{16}$$

$$0 \leq t_i \leq o_i \quad i = 1, \ldots, n. \tag{17}$$

Again, the details of the solution of this concave optimization problem are given in Section 7.

4 Evaluation and Comparison with Other Approaches

We showed through the example in Section 2 that the reward accrued by *any* Mandatory-First scheme [7] may only be approximately half of that of the optimal algorithm. We now prove that, under worst-case scenario, the ratio of the reward accrued by a Mandatory-First approach to the reward of the optimal algorithm may be arbitrarily close to zero. This is a consequence of Theorem 3, which allows us to construct a specific instance of the problem by choosing an arbitrarily large integer r.

Theorem 3. *There is an instance of periodic reward-based scheduling problem where the ratio*

$$\frac{\text{Reward of the best Mandatory First scheme}}{\text{Reward of the optimal scheme}} = \frac{2}{r}$$

for any integer $r \geq 2$.

Proof. Consider two tasks T_1 and T_2 such that $P_2/P_1 = r$, $f_1(t_1) = k_1 t_1$, $f_2(t_2) = k_2 t_2$ and $k_1/k_2 = r(r-1)$. Furthermore, let $m_2 = \frac{1}{2}(r\, o_2)$ and

$$P_1 = m_1 + o_1 + \frac{m_2}{r} = m_1 + \frac{m_2}{r-1},$$

which implies that $o_1 = \frac{m_2}{r(r-1)}$.

This setting suggests that, during any period of T_1, a scheduler has the choice of executing (parts of) O_1 and/or M_2, in addition to M_1.

Note that, under any Mandatory-First policy, during the hyperperiod $[0, P_2]$, the processor will be continuously busy executing mandatory parts until $t = P_2 - P_1 + m_1$. Furthermore, the best algorithm among Mandatory-First policies should use the remaining idle times by scheduling O_1 entirely (since $k_1 > k_2$) and $t_2 = \frac{m_2}{r} = \frac{o_2}{2}$ units of O_2. The resulting schedule is shown in Fig. 3.

The average reward that the best mandatory-first algorithm (MFA) can accrue is therefore:

$$R_{MFA} = \frac{f_1(o_1)}{r} + f_2(t_2).$$

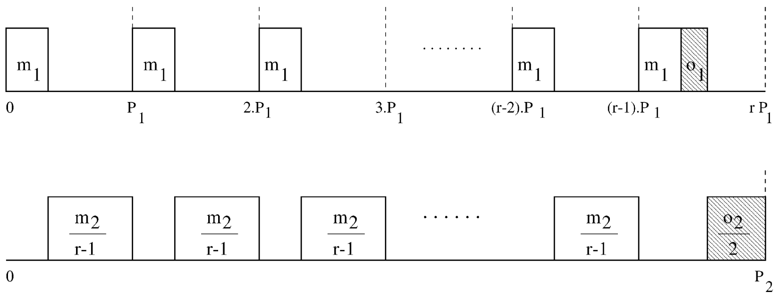


Fig. 3. A schedule produced by the Mandatory-First Algorithm.

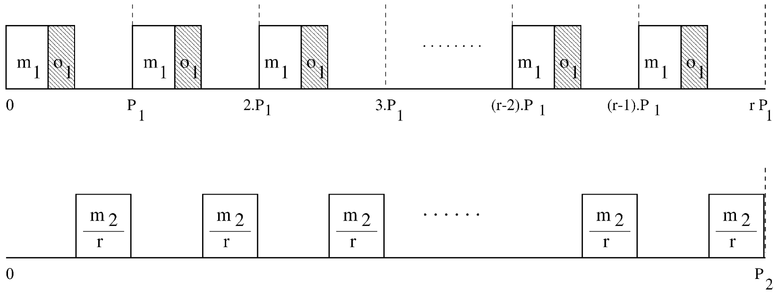


Fig. 4. An optimal schedule.

However, an optimal algorithm (shown in Fig. 4) would choose delaying the execution of M_2 for o_1 units of time, at *every* period of T_1. By doing so, it would have the opportunity of accruing the reward of O_1 at every instance.

The total reward of the above schedule is:

$$R_{OPT} = \frac{r\, f_1(o_1)}{r} = f_1(o_1).$$

The ratio of rewards for the two policies turns out to be (for any $r \geq 2$)

$$\frac{R_{MFA}}{R_{OPT}} = \frac{1}{r} + \frac{f_2(t_2)}{f_1(o_1)} = \frac{1}{r} + \frac{1}{r(r-1)}\frac{m_2}{r}\frac{r(r-1)}{m_2} = \frac{2}{r},$$

which can be made as close as possible to 0 by appropriately choosing r. □

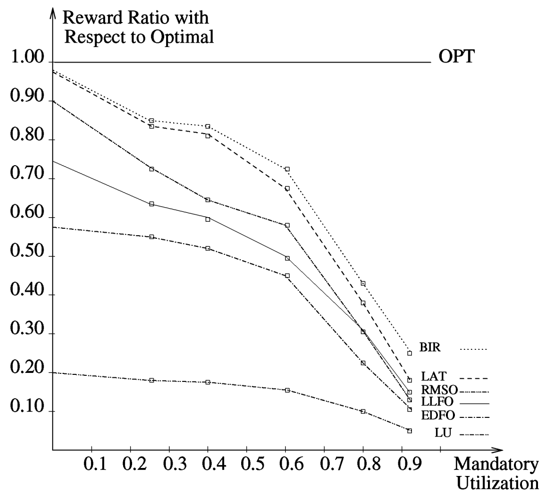
Theorem 3 gives the worst-case performance ratio of Mandatory-First schemes. We also ran simulations using a synthetic task set to investigate the relative performance of Mandatory-First schemes proposed in [7] with different types of reward functions and different mandatory/optional workload ratios.

The Mandatory-First schemes differ by the policy according to which optional parts are scheduled when there is no mandatory part ready to execute. *Rate-Monotonic (RMSO)* and *Least-Utilization (LU)* schemes assign statically higher priorities to *optional parts* with smaller periods and least utilizations, respectively. Among dynamic priority schemes are *Earliest-Deadline-First (EDFO)* and *Least-Laxity-First (LLFO)*, which consider the deadline and laxity of optional parts when assigning priorities. *Least Attained Time (LAT)* aims at balancing execution times of optional parts that are ready by dispatching the one that executed *least* so far. Finally, *Best Incremental Return (BIR)* is an on-line policy which chooses the optional task contributing most to the total reward at a given *slot*. In other words, *at every slot* BIR selects the optional part O_{ij} such that the difference $f_i(t_{ij}+\Delta) - f_i(t_{ij})$ is the largest (here, t_{ij} is the optional service time O_{ij} has already received and Δ is the minimum time slot that the scheduler assigns to any optional task). However, it is still a suboptimal policy since it does not consider the laxity information. The authors indicate in [7] that BIR is too computationally complex to be actually implemented. However, since the total reward accrued by BIR is usually much higher than the other five policies, BIR is used as a yardstick for measuring the performance of other algorithms. We have used a synthetic task set comprised of 11 tasks whose total (mandatory + optional) utilization is 2.3. Individual task utilizations vary from 0.03 to 0.6. We implemented all the MFA schemes mentioned above and generated the schedule during the hyperperiod. Considering exponential, logarithmic, and linear reward functions as separate cases, we measured the reward ratio of six Mandatory-First schemes with respect to our optimal algorithm. The tasks' characteristics (including reward functions) are given in Table 1. In our experiments, we first set mandatory utilization to 0 (which corresponds to the case of all-optional workload), then increased it to 0.25, 0.4, 0.6, 0.8, and 0.91 subsequently.

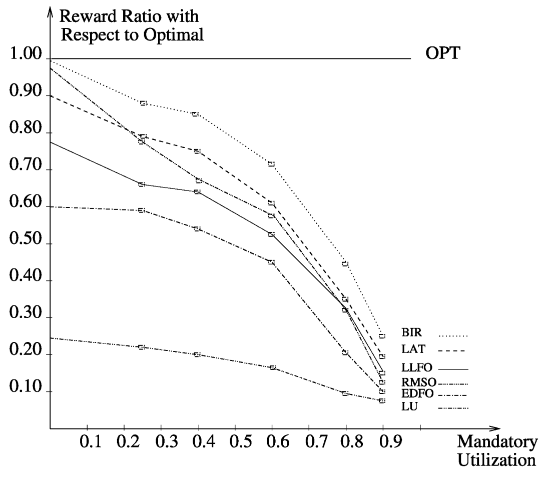
Figs. 5 and 6 show the reward ratio of six Mandatory-First schemes with respect to our optimal algorithm as a function of mandatory utilization for different types of reward functions. A common pattern appears: The optimal algorithm improves more dramatically with the increase in mandatory utilization. The other schemes miss the opportunities of executing "valuable" optional parts while constantly favoring mandatory parts. The reward loss becomes striking as the mandatory workload increases. Figs. 5a and 5b show the reward ratio for the case of exponential and logarithmic reward functions, respectively.

TABLE 1
Task Characteristics

Task id	P_i	$m_i + o_i$	$f_i^1(t)$	$f_i^2(t)$	$f_i^3(t)$
1	20	10	$15\,(1-e^{-t})$	$7\,\ln(20\,t+1)$	$5\,t$
2	30	18	$20\,(1-e^{-3t})$	$10\,\ln(50\,t+1)$	$7\,t$
3	40	5	$4\,(1-e^{-t})$	$2\,\ln(10\,t+1)$	$2\,t$
4	60	2	$10\,(1-e^{-0.5t})$	$5\,\ln(5\,t+1)$	$4\,t$
5	60	2	$10\,(1-e^{-0.2t})$	$5\,\ln(25\,t+1)$	$4\,t$
6	80	12	$5\,(1-e^{-t})$	$3\,\ln(30\,t+1)$	$2\,t$
7	90	18	$17\,(1-e^{-t})$	$8\,\ln(8\,t+1)$	$6\,t$
8	120	15	$8\,(1-e^{-t})$	$4\,\ln(6\,t+1)$	$3\,t$
9	240	28	$8\,(1-e^{-t})$	$4\,\ln(9\,t+1)$	$3\,t$
10	270	60	$12\,(1-e^{-0.5t})$	$6\,\ln(12\,t+1)$	$5\,t$
11	2160	300	$5\,(1-e^{-t})$	$3\,\ln(15\,t+1)$	$2\,t$



(a)



(b)

Fig. 5. The reward ratio of Mandatory-First schemes: strictly concave reward functions (a) exponential, (b) logarithmic functions.

The curves for these strictly concave reward functions are fairly similar: BIR performs best among Mandatory-First schemes and its performance decreases as the mandatory utilization increases; for instance, the ratio falls below 0.75 (for both curves) when mandatory utilization is 0.6. Other algorithms which are more amenable to practical implementations (in terms of runtime overhead) than BIR perform even worse. However, it is worth noting that the performance of LAT is close to that of BIR. This is to be expected since task sets with strictly concave reward functions usually benefit from "balanced" optional service times. Finally, at high mandatory utilizations, the CPU time that can be used for optional executions is naturally diminished, which results in a convergence of all MFA schemes for strictly concave reward functions: The total reward is only slightly affected by the choice of optional task to dispatch.

Fig. 6 shows the reward ratio for linear reward functions. Although the reward ratio of Mandatory-First schemes again decreases with the mandatory utilization, the decrease is less dramatic than in the case of concave functions (see above). However, note that the ratio is typically less than 0.5 for the five practical schemes. It is interesting to observe that the reward of (impractical) BIR method now remains comparable to that of optimal, even in the higher mandatory utilizations: The difference is less than 15 percent. In our opinion, the main reason for this behavior change lies in the fact that, for a given task, the reward of optional execution slots in different instances does not make a difference in the linear case. In contrast, not executing the "valuable" *first slot(s)* of a given instance creates a tremendous effect for nonlinear concave functions. The improvement of the optimal algorithm would be larger for a larger range of k_i values (where k_i is the coefficient of the linear reward function). We note that the worst-case performance of BIR may be arbitrarily bad with respect to the optimal one for linear functions, as Theorem 3 suggests.

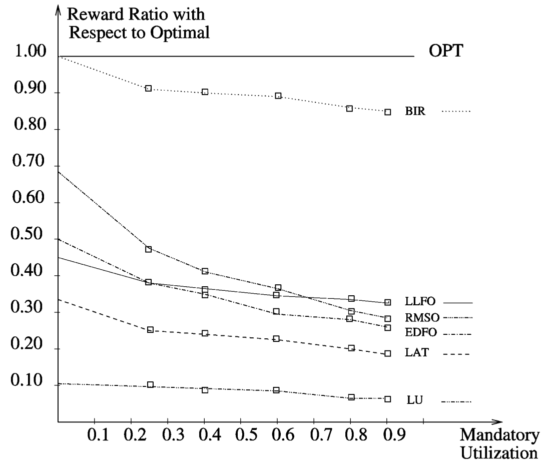


Fig. 6. The reward ratio of Mandatory-First schemes: linear reward functions.

5 Limits on the Optimality of Identical Service Times

We underline that Theorem 1 was the key to eliminate a potentially exponential number of unknowns (t_{ij} values) and thereby to obtain an optimization problem of n variables. One is naturally tempted to ask whether the optimality of identical service times is *still* preserved if some fundamental assumptions of the model are relaxed. Unfortunately, attempts to reach further optimality results for extended/different models remain inconclusive, as the following propositions suggest.

Proposition 1. *The optimality of identical service times no longer holds if the* Deadline = Period *assumption is relaxed.*

Proof. We will prove the statement by providing a counter-example. Assume that we allow the deadline of a task to be less than its period. Consider the two tasks shown in Table 2.

TABLE 2
Two-Task System

id	m_i	o_i	P_i	$F_i(t)$
1	3	3	8	t
2	7	5	16	t

Further, assume that the deadline of T_2 is $D_2 = 10 < P_2$, while the deadline of T_1 coincides with its period, (i.e., $d_1 = 8$). Note that the tight deadline of T_2 makes it impossible to schedule any optional part until $t = 13$, after which we are able to schedule O_1 for three units. This optimal schedule is shown in Fig. 7. On the other hand, if one commits to identical service times per instance, it is clear that we may not schedule *any* optional part since we could not execute O_1 in the first instance of T_1 (Fig. 8). □

Next, suppose that the deadlines are equal to the periods, but we have to adopt a static priority scheduling policy. It was already mentioned in Section 3 that if the periods are harmonic, then we can use RMS without compromising optimality. But, in the general case where the periods are not necessarily harmonic, this is not true *even if we are investigating the "best" schedule within the context of a given static priority assignment.*

Proposition 2. *In the general case, the optimality of identical service times no longer holds if we commit to a static priority assignment.*

Proof. Again, to prove the statement with a counter-example, consider the task set shown in Table 3.

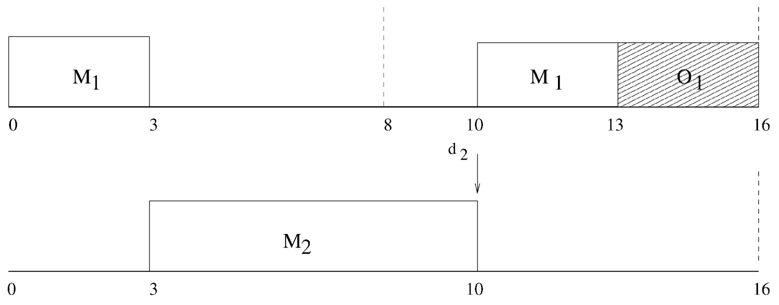


Fig. 7. The optimal schedule for the task system in Table 2.

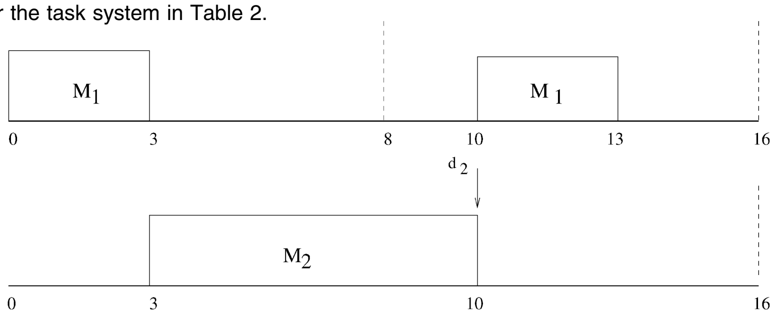


Fig. 8. Best schedule with identical service times for the task system in Table 2.

TABLE 3
Two-Task System

id	m_i	o_i	P_i	$F_i(t)$
1	2	4	6	t
2	3	0	8	t

TABLE 4
Two-Task System

id	m_i	o_i	P_i	$F_i(t)$
1	0	8	8	t^2
2	0	6	16	$2\,t^2$

As we have only two tasks, we will consider the cases where T_1 or T_2 have higher priority and show that, in every case, the reward of the optimal schedule differs from the one obtained with identical service times per instance assumption.

Case 1—T_1 has higher priority: It is easy to see that we can construct a schedule which *fully* utilizes the timeline during the interval $[0, lcm(6,8) = 24]$ (Fig. 9a). This schedule is also immediately optimal since the reward function is linear (observe that $o_2 = 0$ and, hence, we do not receive any reward for executing O_2). But, we remark that we cannot execute O_1 for more than one unit on its first instance in *any* feasible schedule—without violating the deadline of T_2. Therefore, we would have ended up with a lower reward after executing one unit of O_1 at each instance if we had committed to identical service times (Fig. 9b).

Case 2—T_2 has higher priority: The optimality is similarly compromised if T_2 has higher priority. While the optimal schedule (Fig. 10a) fully utilizes the timeline, the best schedule with $t_{1i} = 1 \ (i = 1, \ldots, 4)$ (Fig. 10b) remains suboptimal. □

We remark that Proposition 2 also has implications for the Q-RAM model [25] since it points to the impossibility of achieving optimality with identical service times by using a static priority assignment.

The final proposition in this section illustrates the fact that the optimality of identical service times is also sensitive to the concavity of reward functions.

Proposition 3. *The optimality of identical service times no longer holds if the concavity assumption about the reward functions is relaxed.*

Proof. Consider two harmonic tasks without mandatory parts whose parameters are given in Table 4.

Note that t_2 should be assigned its maximum possible value (i.e., the upper bound 6) since the marginal return of F_2 is larger than F_1 everywhere. An optimal schedule maximizing the average reward for these two tasks is depicted in Fig. 11.

The sum of the average rewards in the optimal schedule is $2.6^2 + \frac{2^2+8^2}{2} = 106$. However, if we commit ourselves to the equal service times per instances, we can find no better schedule than the one shown in Fig. 12, whose reward is only $5^2 + 2 \cdot 6^2 = 97$.

It is not difficult to construct a similar example for the tasks with 0-1 constraints as well, which implies that even *the number* of variables to deal with (the t_{ij}s) may be prohibitively large in these problems. □

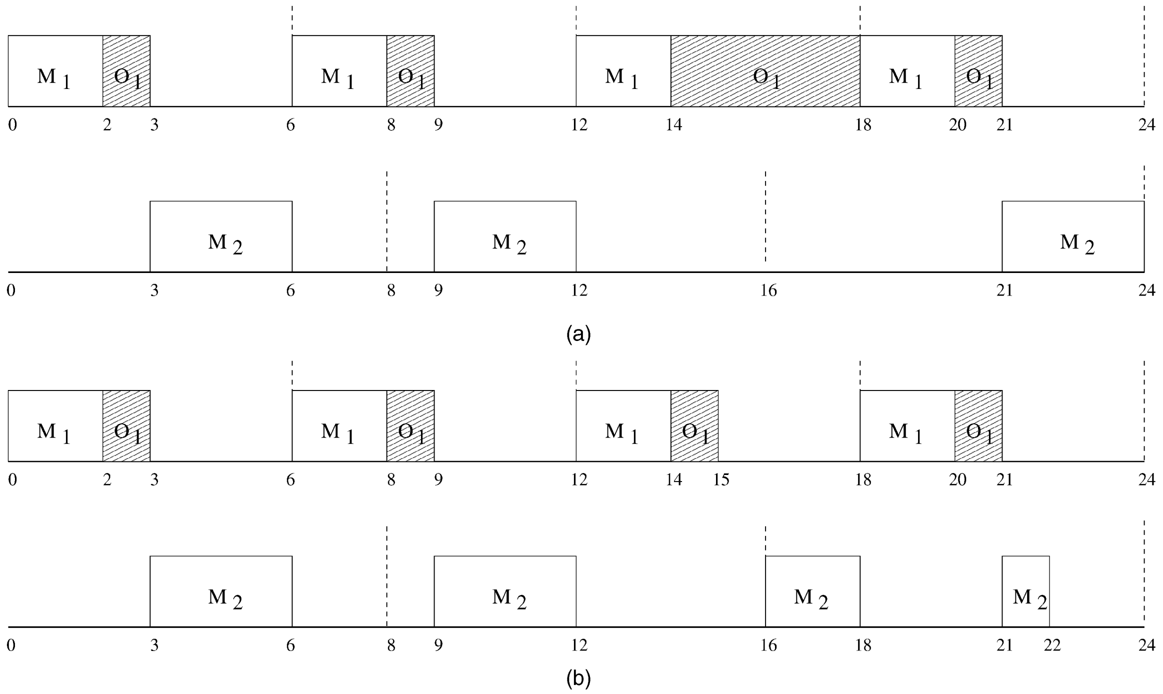
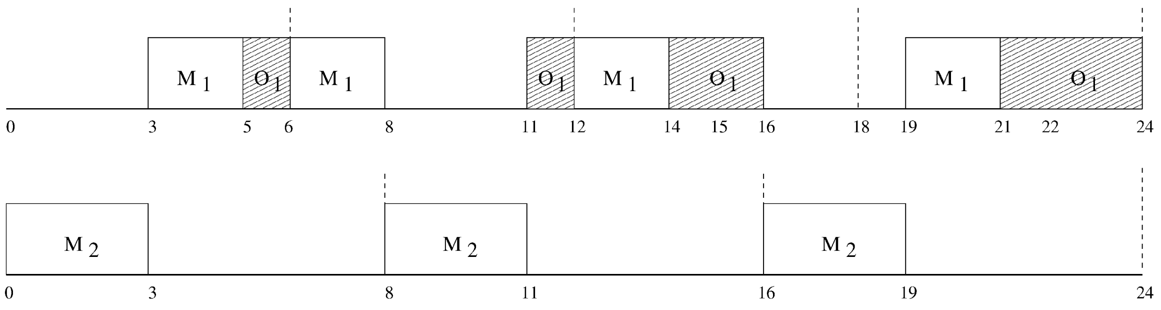


Fig. 9. Case 1: T_1 has the higher priority: (a) the optimal schedule, (b) the best schedule with identical service times.


(a)

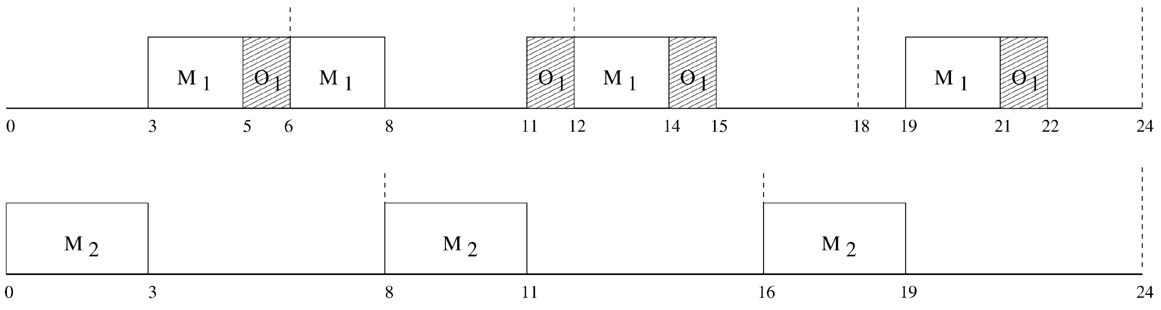

(b)

Fig. 10. Case 2: T_2 has the higher priority: (a) the optimal schedule, (b) the best schedule with identical service times.

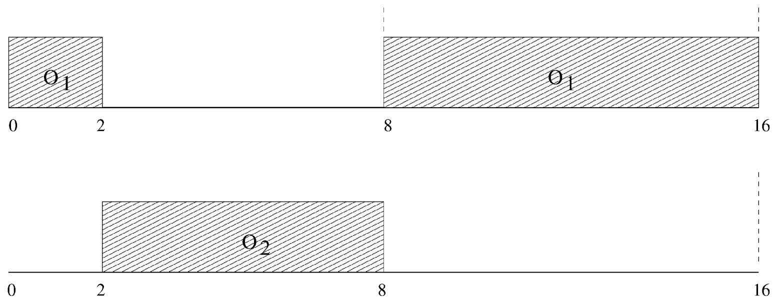


Fig. 11. An optimal schedule for the task system in Table 4.

6 Intractability of Convex Reward-Based Scheduling Problem

As we mentioned before, maximizing the total (or average) reward with $0/1$ constraints case had already been proven to be NP-Complete in [21]. Similarly, in Section 5, we showed that, if the reward functions are convex, the optimality of identical service times is not preserved. In this section, we show that, in fact, convex reward functions result in an NP-Hard problem, even with identical periods.[2]

We now show how to transform the SUBSET-SUM problem, which is known to be NP-Complete, to REW-PER with convex reward functions.

SUBSET-SUM: Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of positive integers and the integer M, is there a set $S_A \subseteq S$ such that $\sum_{s_i \in S_A} s_i = M$?

2. We underline that, as of this writing, most of the applications we investigated have nonincreasing marginal returns, hence they can be represented best by concave reward functions. However, we include the convex reward functions' case and show its intractability in order to complete the theoretical investigation of the periodic reward-based scheduling problem.

We construct the corresponding REW-PER instance as follows: Let $W = \sum_{i=1}^{n} s_i$. Now, consider a set of n periodic tasks with the same period M and mandatory parts $m_i = 0 \ \forall i$. The reward function associated with T_i is given by:

$$R_i(t_i) = \begin{cases} f_i(t_i) & if \quad 0 \le t \le o_i = s_i \\ f_i(o_i) & if \quad t > o_i = s_i, \end{cases}$$

where $f_i(t_i) = t_i^2 + (W - s_i)t_i$ is a strictly convex and increasing function on nonnegative real numbers.

Notice that $f_i(t_i)$ can be rewritten as $t_i(t_i - s_i) + W t_i$. Also, we underline that having the same periods for all tasks implies that REW-PER can be formulated as:

$$\text{maximize} \quad \sum_{i=1}^{n} t_i \, (t_i - s_i) + W \sum_{i=1}^{n} t_i \qquad (18)$$

$$\text{subject to} \quad \sum_{i=1}^{n} t_i \le M \qquad (19)$$

$$0 \le t_i \le s_i. \qquad (20)$$

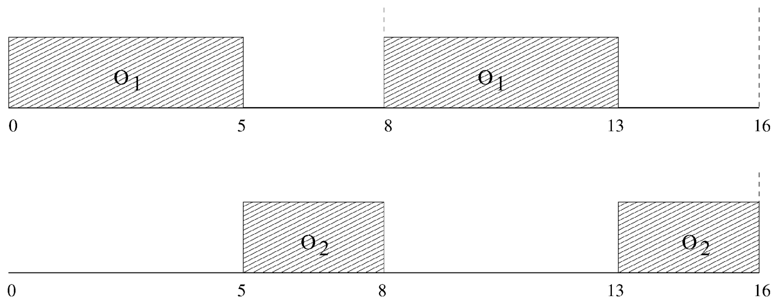


Fig. 12. Best schedule with identical service times for the task system in Table 4.

Let us denote by $MaxRew$ the total reward of the optimal schedule. Observe that, for $0 < t_i < s_i$, the quantity $t_i\,(t_i - s_i) < 0$. Otherwise, at either of the boundary values 0 or s_i, $t_i\,(t_i - s_i) = 0$. Hence, $MaxRew \leq WM$.

Now, consider the question: "Is $MaxRew$ equal to WM?" Clearly, this question can be answered quickly if there is a polynomial-time algorithm for REW-PER where reward functions are allowed to be convex. Furthermore, the answer can be positive only when $\sum_{i=1}^{n} t_i = M$ *and* each t_i is equal to either 0 or s_i. Therefore, $MaxRew$ is equal to WM if and only if there is a set $S_A \subseteq S$ such that $\sum_{s_i \in S_A} s_i = M$, which implies that REW-PER with convex reward functions is NP-Hard.

7 Solution of Periodic Concave Reward-Based Scheduling Problem

Corollaries 1 and 2 reveal that the two optimization problems whose solutions provide optimal service times for uniprocessor and multiprocessor systems share a *common form*:

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{n} f_i(t_i) \\ &\text{subject to} && \sum_{i=1}^{n} b_i\, t_i \leq d \\ & && t_i \leq o_i \quad i = 1, 2, \ldots, n \\ & && 0 \leq t_i \quad i = 1, 2, \ldots, n, \end{aligned}$$

where d (the "slack" available for optional executions) and $b_1, b_2, \ldots, b_n$ are positive rational numbers. In this section, we present polynomial-time solutions for this problem, where each f_i is a nondecreasing, concave, and differentiable[3] function.

First note that, if the available slack is large enough to accommodate every optional part entirely (i.e., if $\sum_{i=1}^{n} b_i\, o_i \leq d$), then the choice $t_i = o_i \,\forall\, i$ clearly maximizes the objective function due to the nondecreasing nature of reward functions.

Otherwise, the slack d should be used in its entirety since the total reward never decreases by doing so (again due to the nondecreasing nature of the reward functions). In this case, we obtain a concave optimization problem with lower and upper bounds, denoted by OPT-LU. An instance of OPT-LU is specified by the set of nondecreasing concave functions $\mathbf{F} = \{f_1, \ldots, f_n\}$, the set of upper bounds $\mathbf{O} = \{o_1, \ldots, o_n\}$, and the available slack d. The aim is to:

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(t_i) \tag{21}$$

$$\text{subject to} \quad \sum_{i=1}^{n} b_i\, t_i = d \tag{22}$$

$$t_i \leq o_i \quad i = 1, 2 \ldots, n \tag{23}$$

$$0 \leq t_i \quad i = 1, 2, \ldots, n, \tag{24}$$

where $0 < d < \sum_{i=1}^{n} b_i \cdot o_i$.

Special Case of Linear functions: We address separately the case when $\mathbf{F}$ comprises solely linear functions since the time complexity can be considerably reduced by using this information. Note that, for a function $f_i(t_i) = k_i \cdot t_i$, if we increase t_i by Δ, then total reward increases by $k_i\,\Delta$. However, by doing so, we make use of $b_i\,\Delta$ units of slack (d is reduced by $b_i\,\Delta$ due to (22)). Hence, the "marginal return" of task T_i per slack unit is $w_i = \frac{k_i}{b_i}$. Now, consider another function $f_j(t_j) = k_j \cdot t_j$. Clearly, $w_j = \frac{k_j}{b_j}$. If $w_j > w_i$, then task T_j *should be always favored* with respect to T_i since the marginal return of f_j is strictly greater than f_i *everywhere*. Repeating the argument for every pair of tasks, we can obtain the following optimal strategy.

We first order the functions according to their marginal returns $w_i = \frac{k_i}{b_i}\;\; i = 1, 2, \ldots, n$. Let f_1 be the function with the largest marginal return, f_2 the second, and so on (ties are broken arbitrarily). If $b_1\, o_1 \geq d$, then we set $t_1 = d/b_1$ and we are done (we are using the entire slack for T_1). If $b_1\, o_1 < d$, then we set $t_1 = o_1$ and d is reduced accordingly ($d = d - b_1\, o_1$). Next, we repeat the same step for the next "most valuable" function, f_2. After at most n iterations, the slack d is completely consumed. We note that this solution for linear functions is analogous to the one presented in [25]. The dominant factor in the time complexity comes from the initial sorting procedure, hence, in the special case of all-linear functions, OPT-LU can be solved in time $O(n \log n)$.

When $\mathbf{F}$ contains nonlinear functions, the procedure becomes more involved. In the next two subsections, we introduce two auxiliary optimization problems, namely Problem OPT (which considers only the equality constraint) and Problem OPT-L (which considers only

3. In the auxiliary optimization problems which will be introduced shortly, the differentiability assumption holds as well.

the equality and lower bound constraints), which will be used to solve OPT-LU.

7.1 Problem OPT: Case of the Equality Constraint

An instance of the problem OPT is characterized by the set $\mathbf{F} = \{f_1, \ldots, f_n\}$ of nondecreasing concave functions and the slack d:

$$\begin{aligned} \text{maximize} \qquad & \sum_{i=1}^{n} f_i(t_i) \\ \text{subject to} \qquad & \sum_{i=1}^{n} b_i \; t_i = d. \end{aligned}$$

As can be seen, OPT does not take into account the lower and upper bound constraints of Problem OPT-LU. The algorithm which returns the solution of Problem OPT is denoted by "Algorithm OPT."

When $\mathbf{F}$ is composed solely of *nonlinear* reward functions, the application of Lagrange multipliers technique [22] to the Problem OPT yields:

$$\frac{1}{b_i} f_i'(t_i) - \lambda = 0 \qquad i = 1, 2, \ldots, n, \tag{25}$$

where λ is the common Lagrange multiplier and $f_i'(t_i)$ is the derivative of the reward function f_i. The quantity $\frac{1}{b_i} f_i'(t_i)$ in (25) actually represents the *marginal return* contributed by T_i to the total reward, which we will denote as $w_i(t_i)$. Observe that, since $f_i(t_i)$ is nondecreasing and concave by assumption, both $w_i(t_i)$ and $f_i'(t_i)$ are nonincreasing and positive valued. Equation (25) implies that the marginal returns $w_i(t_i) = \frac{1}{b_i} f_i'(t_i)$ *should* be equal for all reward functions in the optimal solution $\{t_1, \ldots, t_n\}$. Considering that the equality $\sum_{i=1}^{n} b_i \; t_i = d$ should also hold, one can obtain closed formulas in most of the cases which occur in practice. The closed formulas presented below are obtained by this method.

- For *logarithmic* reward functions of the form $f_i(t_i) = \ln(k_i \cdot t_i + c_i)$,

$$t_1 = \frac{d + \sum\limits_{j=1}^{n} \frac{c_j}{k_j} - \frac{c_1}{k_1 \, b_1} \sum\limits_{j=1}^{n} b_j}{\frac{1}{b_1} \sum\limits_{j=1}^{n} b_j}$$

  and $t_j = b_1 \, t_1 + \frac{c_1}{k_1 \, b_1} - \frac{c_j}{k_j \, b_j} \;\; \forall j \;\; 1 < j \leq n$.

- For *exponential* reward functions of the form $f_i(t_i) = c_i(1 - e^{-k_i t_i})$,

$$t_1 = \frac{d - \sum\limits_{j=1}^{n} \frac{1}{k_j} [\ln (\frac{c_j b_1 k_j}{c_1 b_j k_1})]}{\sum\limits_{j=1}^{n} \frac{k_1}{k_j}}$$

  and $t_j = \frac{1}{k_j} [k_1 \, t_1 + \ln (\frac{c_j b_1 k_j}{c_1 b_j k_1})] \;\; \forall j \;\; 1 < j \leq n$.

- For "kth root" reward functions of the form $f_i(t_i) = c_i \; t_i^{1/k}$ $(k > 1)$,

$$t_1 = \frac{d}{\sum\limits_{j=1}^{n} (\frac{b_j c_1}{b_1 c_j})^{\frac{1}{k-1}}}$$

  and $t_j = t_1 \, (\frac{b_j c_1}{b_1 c_j})^{\frac{1}{k-1}} \;\; \forall j \;\; 1 < j \leq n$.

When it is not possible to find a closed formula, following exactly the approach presented in [14], [15], [18], we solve for λ in the equation $\sum_{i=1}^{n} b_i \; h_i(\lambda) = d$, where $h_i(k)$ is the inverse function of $\frac{1}{b_i} f_i'(t_i) = w_i(t_i)$ (we assume the existence of the derivative's inverse function whenever f_i is nonlinear, complying with [14], [15], [18]). Once λ is determined, $t_i = h_i(\lambda), \; i = 1, \ldots, n$ is the optimal solution.

We now examine the case where $\mathbf{F}$ is a *mix* of linear and nonlinear functions. Consider two linear functions $f_i(t) = k_i \cdot t$ and $f_j(t) = k_j \cdot t$. The marginal return of $f_i(t_i)$ is $w_i(t_i) = \frac{k_i}{b_i} = w_i$ and that of f_j is $w_j(t_j) = \frac{k_j}{b_j} = w_j$. Now, if $w_j > w_i$, then the service time t_i should be definitely zero since marginal return of f_i is strictly less than f_j everywhere. After this elimination process, if there are $p > 1$ linear functions with the same (largest) marginal return w_{max}, then we will consider them as a single linear function in the procedure below and evenly divide the returned service time t_{max} among t_j values corresponding to these p functions. Hence, without loss of generality, we assume that $f_n(t) = k_n \cdot t$ is the only linear function in $\mathbf{F}$. Its marginal return is $w_n(t_n) = \frac{k_n}{b_n} = w_{max}$. We first compute the optimal distribution of slack d among tasks with nonlinear reward functions $f_1, \ldots, f_{n-1}$. By the Lagrange multipliers technique, $w_i(t_i) - \lambda = 0$ and, thus, $w_1(t_1^*) = w_2(t_2^*) = \ldots = w_{n-1}(t_{n-1}^*) = \lambda$ at the optimal solution $t_1^*, t_2^*, \ldots, t_{n-1}^*$.

Now, we distinguish two cases:

- $\lambda >= w_{max}$. In this case, $t_1^*, t_2^*, \ldots, t_{n-1}^*$ and $t_n = 0$ is the optimal solution to OPT. To see this, first remember that all the reward functions are concave and nondecreasing, hence $w_i(t_i^* - \epsilon) \geq w_i(t_i^*) \geq w_n(\epsilon) = w_{max} \; i = 1, \ldots, n-1$ for all $\epsilon >= 0$. This implies that transferring some service time from another task T_i to T_n would mean favoring the task which has the smaller marginal reward rate and would not be optimal.
- $\lambda < w_{max}$. In this case, reserving the slack d solely to tasks with nonlinear reward functions means violating the best marginal rate principle and, hence, is not optimal. Therefore, we should decrease service time t_i until $w_i(t_i)$ reaches the level of w_{max} for $i = 1, 2, \ldots, n-1$, *but not beyond*. Solving $h_i(w_{max}) = t_i$ for $i = 1, 2, \ldots, n-1$ and assigning any remaining slack $\frac{d - \sum_{i=1}^{n-1} t_i}{b_n}$ to t_n (the service time of unique task with linear reward function) clearly satisfies the best marginal rate principle and achieves optimality.

7.2 Problem OPT-L: Case of Lower Bounds

In this section, we present a solution for problem OPT-L and we improve on this solution in Section 7.2.1. An instance of Problem OPT-L is characterized by the set

$\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$ of nondecreasing concave reward functions, and the available slack d:

$$\text{maximize} \qquad \sum_{i=1}^{n} f_i(t_i) \tag{26}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} b_i \, t_i = d \tag{27}$$

$$0 \leq t_i \quad i = 1, 2, \ldots, n. \tag{28}$$

To solve OPT-L, we first evaluate the solution set S_{OPT} to the corresponding problem OPT and check whether all inequality constraints are automatically satisfied. If this is the case, the solution set S_{OPT-L} of Problem OPT-L is clearly the solution S_{OPT}. Otherwise, we will construct S_{OPT-L} iteratively as described below.

A well-known result of nonlinear optimization theory states that the solution S_{OPT-L} of Problem OPT-L should satisfy so-called *Kuhn-Tucker conditions* [22], [24]. Furthermore, Kuhn-Tucker conditions are also sufficient in the case of concave reward functions [22], [24]. For Problem OPT-L, Kuhn-Tucker conditions comprise (27), (28), and:

$$-\frac{1}{b_i} f_i'(t_i) + \lambda - \frac{\mu_i}{b_i} = 0 \quad i = 1, 2 \ldots n \tag{29}$$

$$-\mu_i \, t_i = 0 \quad i = 1, 2 \ldots n \tag{30}$$

$$\mu_i \geq 0 \quad i = 1, 2 \ldots n, \tag{31}$$

where $\lambda, \mu_1, \mu_2, \ldots, \mu_n$ are Lagrange multipliers. The necessary and sufficient character of Kuhn-Tucker conditions indicates that any $2n+1$ tuple $(t_1, t_2, \ldots, t_n, \mu_1, \mu_2, \ldots, \mu_n, \lambda)$ which satisfies conditions (27) through (31) provides optimal t_i values for OPT-L.

One method of solving the optimization problem OPT-L is to find a solution to the $2n+1$equations (27), (29), and (30) which satisfies constraint sets (28) and (31). Iteratively solving the $2n+1$ nonlinear equations is a complex process which is not guaranteed to converge. In this paper, we follow a different approach. Namely, we use the Kuhn-Tucker conditions (29), (30), and (31) to prove some useful properties of the optimal solution. Our method is based on carefully using the properties that we derive in order to *refine* the solution of the optimization problem OPT.

Claim 2. *If S_{OPT} violates some inequality constraints given by (28), then $\exists i \; \mu_i > 0$.*

Proof. Assume to the contrary that $\forall i \; \mu_i = 0$. In this case, Kuhn-Tucker conditions reduce to the equality constraint (27), the set of inequality constraints (28), plus the Lagrangian condition given in (25). On the other hand, S_{OPT}, which is the solution of OPT, should satisfy (27) and the Lagrangian condition (25). In other words, solving OPT is always equivalent to solving a set of nonlinear equations which are identical to Kuhn-Tucker conditions of OPT-L *except inequality constraints, by setting* $\forall i \; \mu_i = 0$. Hence, if there were a solution to OPT-L where $\forall i \; \mu_i = 0$, then that solution would be returned by the algorithm solving OPT *and* would not violate inequality constraints. However, given that the solution S_{OPT} failed to satisfy all the inequality constraints, we reach a contradiction. Therefore, there exists at least one Lagrange multiplier μ_i which is strictly greater than 0. □

Claim 3. $\exists j \; \mu_j = 0$.

Proof. For the sake of contradiction, assume that $\forall j \; \mu_j > 0$. In this case, (30) enforces that $\forall i \; t_i = 0$. If this were true, $\sum_{i=1}^{n} b_i \, t_i$ would be equal to 0, leaving the slack d totally unutilized. In this case, this clearly would not be the optimal solution due to the nondecreasing nature of reward functions. □

In the rest of the paper, we use the expression "the set of functions" instead of "the set of **indices** of functions" unless confusion arises. Let:

$$\Pi = \{x | \frac{1}{b_x} f_x'(0) \leq \frac{1}{b_i} f_i'(0) \; \forall i\}. \tag{32}$$

Remember that $\frac{1}{b_x} f_x'(t_x)$ is the marginal return associated with $f_x(t_x)$ and is denoted by $w_x(t_x)$. Informally, Π contains the functions $f_x \in \mathbf{F}$ with the smallest marginal returns at the lower bound 0, $w_x(0)$.

Lemma 1. *If S_{OPT} violates some inequality constraints, then, in S_{OPT-L}, $t_m = 0 \; \forall m \in \Pi$.*

Proof. Assume that $\exists m \in \Pi$ such that $t_m > 0$. In this case, (30) implies that the corresponding $\mu_m = 0$. By Claim 2, we know that $\exists j$ such that $\mu_j > 0$. By (30), $t_j = 0$. Using (29), we can write

$$\frac{1}{b_m} f_m'(t_m) - \frac{1}{b_j} f_j'(0) = \frac{\mu_j}{b_j} > 0.$$

Since $t_m > 0$, the concavity property of f_m suggests that $\frac{1}{b_m} f_m'(t_m) \leq \frac{1}{b_m} f_m'(0)$. But, in this case, we obtain $\frac{1}{b_m} f_m'(0) - \frac{1}{b_j} f_j'(0) \geq \frac{\mu_j}{b_j} > 0$, contradicting the assumption that $m \in \Pi$. Hence, $\mu_m > 0$ and, by (31), $t_m = 0$. □

In view of Lemma 1, in Fig. 13 we present the algorithm to solve Problem OPT-L.

Complexity: The time complexity $C_{OPT}(n)$ of Algorithm OPT is $O(n)$: If the mentioned closed formulas apply, then the complexity is clearly linear; otherwise, the unique unknown λ can be solved in linear time under concavity assumptions, as indicated in [14], [15], [18]. Lemma 1 immediately implies the existence of an algorithm which sets $t_m = 0 \; \forall m \in \Pi$, and then reinvokes Algorithm OPT for the remaining tasks and slack (in case some inequality constraints are violated by S_{OPT}). Since the number of invocations is bounded by n, the complexity of the algorithm which solves OPT-LU is $O(n^2)$.

7.2.1 Faster Solution for Problem OPT-L

In this section, we present a faster algorithm of time complexity $O(n \cdot \log n)$ to solve OPT-L. We will make use of the new (faster) algorithm in the final solution of OPT-LU.

Consider Algorithm OPT-L depicted in Fig. 13. Let F_k be the set of functions passed to OPT during the kth iteration of Algorithm OPT-L, and Π_k be the set of functions with minimum marginal returns at the lower bounds (minimum $w_i(0)$ values) during the kth iteration

```
Algorithm OPT-L(F,d)
1  Evaluate the solution $S_{OPT}$ of the optimization problem
     (without inequality constraints)
2  If all the inequality constraints are satisfied then $S_{OPT-L}$=$S_{OPT}$; exit
3  Compute $\mathbf{\Pi}$ from equation 32
4  Set $t_m = 0\ \forall m\ \in \mathbf{\Pi}$
5  Set $\mathbf{F} =\ \mathbf{F} - \mathbf{\Pi}$
6  goto Step 1
```

Fig. 13. Algorithm to solve Problem OPT-L.

(formally, $\Pi_k = \{x | \frac{1}{b_x} f'_x(0) \leq \frac{1}{b_y} f'_y(0)\ \forall y | y \in F_k\}$). Also let $n^* \leq n$ be the number of *distinct* $w_i(0)$ values among functions in $\mathbf{F}$ and $m^* \leq n^*$ be the iteration number where Algorithm OPT returns a solution set which satisfies the constraint set given by (28) for the remaining t_i values. Note that the elements of $\Pi_i\ \ i = 1, \ldots, n^*$ can be produced in $O(n \cdot \log n)$ time during a preprocessing phase. Clearly, Algorithm OPT-L sequentially sets $t_i = 0\ \forall i\ \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_{m^*-1}$ until Algorithm OPT returns a solution which does not violate the constraint set for the remaining unknowns at the (m^*)th iteration.

A tempting idea is to use binary search in the range $[1, n^*]$ to locate the critical index m^* in a faster way. However, to justify the correctness of such a procedure, one needs to prove that if one had further set $t_i = 0\ \forall i\ \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_y$, where $y \geq m^*$, and subsequently invoked the algorithm OPT, then S_{OPT} obtained in this way would have still satisfied the constraint set given by (28). Notice that if this property does not hold, then it is not possible to determine the "direction" of the search by simply testing S_{OPT} at a given index i since we must be assured that there exists a *unique* index m^* such that:

- Setting $t_i = 0\ \forall i \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_y$ and invoking OPT *does not provide* a solution S_{OPT} which satisfies the inequality constraints whenever $1 \leq y < m^* - 1$.
- Setting $t_i = 0\ \forall i \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_y$ and invoking OPT *does provide* a solution S_{OPT} which satisfies the inequality constraints whenever $m^* - 1 \leq y \leq n$.

The first of these properties follows directly from the correctness of Algorithm OPT-L. It turns out that the second property also holds for *concave* objective functions, as proven below. Hence, the time complexity $C_{OPT-L}(n)$ may be reduced to $O(n \cdot \log n)$ by using a binary search-like technique. Algorithm FAST-L, which solves Problem OPT-L in time $O(n \cdot \log n)$, is shown in Fig. 14.

7.2.2 Correctness Proof of the Fast Algorithm

We begin by introducing the following additional notation regarding the kth iteration of Algorithm OPT-L.

- $t_{i,k}$: service time assigned to the optional part of task T_i by OPT during the kth iteration of Algorithm OPT-L,
- $S_{OPT,k} = \{t_{1,k}, \ldots, t_{n,k}\}$: solution produced by OPT during the kth iteration of Algorithm OPT-L,
- $V_k = \{x | t_{x,k} < 0\}$: the set of indices for which the solution $S_{OPT,k}$ *violates* inequality constraints.

Clearly, Algorithm OPT-L successively sets $t_i = 0\ \forall i\ \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_{m^*-1}$ until Algorithm OPT returns a solution which does not violate any constraints for functions in F_{m^*} at the (m^*)th iteration. Algorithm FAST-L uses binary search to determine the critical index m^* efficiently. The correctness of Algorithm OPT-L assures that $\forall\ y < m^* - 1$, setting $t_i = 0\ \forall i\ \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_y$ and invoking OPT would yield a nonempty violating set V_y for the remaining tasks. Finally, Proposition 4 establishes that $\forall\ y \geq m^* - 1$, V_y will always remain empty after setting $t_i = 0\ \forall i\ \in \Pi_1 \cup \ldots \Pi_{m^*-1} \ldots \cup \Pi_y$ and invoking OPT since this would leave even *more* slack for the remaining tasks. In the algorithm FAST-L, a specific index m is tested at each iteration to check whether it satisfies the property $V_m = \emptyset$ and $V_{m-1} \neq \emptyset$. If this is the case, then $m = m^* - 1$ since there is only one index m^* satisfying this property. However, in the case that $V_m \neq \emptyset$, then we can infer that $m < m^* - 1$ and the next probe is determined in the range (m, n^*). Finally, if both $V_m = \emptyset$ and $V_{m-1} = \emptyset$, then we restrict the search in the range $(0, m - 1)$.

Proposition 4. *Suppose that, during the execution of Algorithm OPT-L, $S_{OPT,k}$ does not violate some inequality constraints (i.e., $V_k = \emptyset$), yet we set $t_j = 0$ for $\forall j\ \in \Pi_k$. Then, the $(k+1)$th invocation of Algorithm OPT for the remaining tasks yields $S_{OPT,k+1}$ such that $t_{i,k+1} \geq t_{i,k}$ for all $i \in\ F_{k+1} = F_k - \Pi_k$ (which implies that V_{k+1} is still empty).*

Proof. Note that $t_{i,k} \geq 0\ \forall\ i \in F_k$ by assumption. Based on the optimality property of subproblems, if the kth invocation of Algorithm OPT yields an optimal solution, it will also generate the *optimal* distribution of $d - \sum_{j \in \Pi_k} b_j\ t_{j,k} = d_1 \leq d$ time units among functions in $F_k - \Pi_k$. However, the $(k+1)$th invocation provides optimal distribution of $d - 0 = d$ among functions in $F_k - \Pi_k$ as well (by setting $t_j = 0$ for $\forall j \in \Pi_k$). Thus, two successive invocations of Algorithm OPT can be written as:

$$\begin{aligned} &\text{maximize} && \sum f_i(t_i) \\ &\text{subject to} && \sum b_i\ t_i = d_1 \qquad i \in F_k - \Pi_k \end{aligned}$$

and

```
Algorithm FAST-L(F,d)
1   Evaluate $S_{OPT}$ of the corresponding Problem OPT
2   If all the constraints are satisfied then $S_{OPT-L}$=$S_{OPT}$; exit
3   Enumerate the functions in $\mathbf{F}$ according to the non-decreasing order
    of $w_j(0)$ values and construct the sets $\Pi_i$  $i = 1 \ldots n^*$
4   $m = r = \lfloor n^*/2 \rfloor$
5   Do forever
6       Evaluate $S_{OPT}$ by invoking OPT($\Pi_{m+1} \cup \Pi_{m+2} \ldots \cup \Pi_{n^*}, d$)
7       if a constraint is violated
8           $m = m + \lfloor r/2 \rfloor$; $r = \lfloor r/2 \rfloor$
9       else {
10          $S_1 = S_{OPT}$
11          Evaluate $S_{OPT}$ by invoking OPT($\Pi_m \cup \Pi_{m+1} \ldots \cup \Pi_{n^*}, d$)
12          if a constraint is violated
13              $S_{OPT-L} = \{t_i = 0 \ \forall i \ \in \Pi_1 \cup \Pi_2 \ldots \cup \Pi_{m+1}\} \cup S_1$; exit
14          else {
15              $m = m - \lfloor r/2 \rfloor$; $r = \lfloor r/2 \rfloor$
16          }
17      }
```

Fig. 14. Fast Algorithm for Problem OPT-L.

$$\begin{aligned} &\text{maximize} && \sum f_i(t_i) \\ &\text{subject to} && \sum b_i\, t_i = d_1 + e \quad i \in F_{k+1} = F_k - \Pi_k, \end{aligned}$$

where $e \geq 0$. Hence, the proof will be complete if we show that $\forall i \in \ F_k - \Pi_k, \ t_{i,k+1} \geq t_{i,k}$.

Any solution $S_{OPT,k}$ should satisfy first-order necessary conditions for Lagrangian [22]:

$$\sum b_i\, t_{i,k} = d \tag{33}$$

$$\frac{1}{b_i} f_i'(t_{i,k}) - \lambda = 0 \qquad \forall i \text{ such that } f_i \in F_k. \tag{34}$$

The necessary conditions (34) give

$$\frac{1}{b_c} f_c'(t_{c,k}) = \frac{1}{b_d} f_d'(t_{d,k}) \qquad \forall c, d \in \ F_k - \Pi_k \tag{35}$$

$$\frac{1}{b_c} f_c'(t_{c,k+1}) = \frac{1}{b_d} f_d'(t_{d,k+1}) \qquad \forall c, d \in \ F_k - \Pi_k. \tag{36}$$

For the sake of contradiction, assume that $\exists w \in F_k - \Pi_k$ such that $t_{w,k+1} < t_{w,k}$. Since $e \geq 0$, there must be some $y \in F_k - \Pi_k$ such that $t_{y,k+1} > t_{y,k}$. We distinguish two cases:

1. f_w is nonlinear, that is, its derivative is *strictly* decreasing. Since f_y is also concave, we can write $\frac{1}{b_w} f_w'(t_{w,k+1}) > \frac{1}{b_w} f_w'(t_{w,k})$ and $\frac{1}{b_y} f_y'(t_{y,k+1}) \leq \frac{1}{b_y} f_y'(t_{y,k})$, which are clearly inconsistent with (35) and (36). This can be easily seen by substituting w for c and y for d in (35) and (36), respectively.
2. f_w is linear, which implies that

$$f_w'(t_{w,k+1}) = f_w'(t_{w,k}) = j.$$

In this case, to satisfy (35) and (36), f_y' should be also linear of the form $f_y(t) = h \cdot t$ such that $h = \frac{b_y\, j}{b_w}$. Hence, two functions have the *same* marginal return $w_w = w_y = \frac{j}{b_w} = \frac{h}{b_y}$ But, remembering our assumption from Section 7.1 that Algorithm OPT treats all linear functions of the same marginal return "fairly" (that is, assigns them the same amount of service time), we reach a contradiction since $t_{w,k}$ was supposed to be greater than $t_{y,k}$. □

Complexity: At most $O(\log n)$ probes are made during binary search and at each probe Algorithm OPT is called twice. Recall that Algorithm OPT has the time complexity $O(n)$. The initial cost of sorting the derivative values is $O(n \log n)$. Hence, the total complexity is $C_{OPT-L}(n) = O(C_{OPT}(n) \cdot \log n + n \cdot \log n)$, which is $O(n \cdot \log n)$.

7.3 Combining All Constraints: Solution of Problem OPT-LU

An instance of Problem OPT-LU is characterized by the set $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$ of nondecreasing, differentiable, and concave reward functions, the set $\mathbf{O} = \{o_1, o_2, \ldots, o_n\}$ of upper bounds on the length of optional execution parts, and available slack d:

```
Algorithm OPT-LU(F,O,d)
1   Set $S_{OPT-LU} = \emptyset$
2   if $\mathbf{F} = \emptyset$ then exit
3   Evaluate $S_{OPT-L}$ by invoking Algorithm FAST-L
4   if all upper bound constraints are satisfied then
        $S_{OPT-LU} = (S_{OPT-LU} \cup S_{OPT-L})$; exit
5   compute $\mathbf{\Gamma}$
6   set $t_q = o_q \, \forall q \in \mathbf{\Gamma}$ in $S_{OPT-LU}$
7   set $d = d - \sum_{x \in \Gamma} b_x \, o_x$
8   set $\mathbf{F}=\mathbf{F}-\mathbf{\Gamma}$
9   set $\mathbf{O}=\mathbf{O}-\{\mathbf{o_x} | \mathbf{x} \in \mathbf{\Gamma}\}$
10  Goto Step 2
```

Fig. 15. Algorithm to solve Problem OPT-LU.

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(t_i) \tag{37}$$

$$\text{subject to} \quad \sum_{i=1}^{n} b_i \, t_i = d \tag{38}$$

$$t_i \leq o_i \quad i = 1, 2, \ldots, n \tag{39}$$

$$0 \leq t_i \quad i = 1, 2, \ldots, n. \tag{40}$$

We recall that $0 < d < \sum_{i=1}^{n} o_i$ and $0 < o_i \, \forall i$ in the specification of OPT-LU.

We first observe the close relationship between the problems OPT-LU and OPT-L. Indeed, OPT-LU has only an additional set of upper bound constraints. It is not difficult to see that if S_{OPT-L} satisfies the constraints given by (39), then the solution S_{OPT-LU} of problem OPT-LU is the same as S_{OPT-L}. However, if an upper bound constraint is violated, then we will construct the solution iteratively in a way analogous to that described in the solution of Problem OPT-L.

Let $\Gamma = \{x | \frac{1}{b_x} f'_x(o_x) \geq \frac{1}{b_i} f'_i(o_i) \, \forall i\}$. Informally, Γ contains the functions $f_x \in \mathbf{F}$ with the largest marginal returns at the upper bounds, $w_x(o_x)$.

The algorithm ALG-OPT-LU (see Fig. 15) which solves the problem OPT-LU is based on the successive invocations of FAST-L. First, we find the solution S_{OPT-L} of the corresponding problem OPT-L. We know that this solution is optimal for the simpler problem which does not take into account upper bounds. If the upper bound constraints are automatically satisfied, then we are done. However, if this is not the case, we set $t_q = o_q \, \forall q \in \Gamma$. Finally, we update the sets **F**, **O** and the slack d before going through the next iteration.

Correctness: Most of the algorithm is self-explanatory in view of the results obtained in previous sections. However, lines 5 and 6 of ALG-OPT-LU require further elaboration. In addition to constraints (38), (39), and (40), the necessary and sufficient Kuhn-Tucker conditions for Problem OPT-LU can be expressed as:

$$-\frac{1}{b_i} f'_i(t_i) + \lambda + \frac{\bar{\mu}_i}{b_i} - \frac{\mu_i}{b_i} = 0 \quad i = 1, 2, \ldots, n \tag{41}$$

$$\bar{\mu}_i(t_i - o_i) = 0 \quad i = 1, 2, \ldots, n \tag{42}$$

$$-\mu_i \, t_i = 0 \quad i = 1, 2, \ldots, n \tag{43}$$

$$\bar{\mu}_i \geq 0 \quad i = 1, 2, \ldots, n \tag{44}$$

$$\mu_i \geq 0 \quad i = 1, 2, \ldots, n, \tag{45}$$

where $\lambda, \bar{\mu}_1, \bar{\mu}_2, \ldots, \bar{\mu}_n, \mu_1, \mu_2, \ldots, \mu_n$ are Lagrange multipliers.

Claim 4. *If S_{OPT-L} violates upper bound constraints given by (39), then $\exists i \; \bar{\mu}_i > 0$.*

Proof. Assume that $\forall i \; \bar{\mu}_i = 0$. In this case, Kuhn-Tucker conditions (42) and (44) vanish. Also, conditions (38), (40), (41), (43), and (45) become exactly identical to the Kuhn-Tucker conditions of Problem **OPT-L**. Thus, S_{OPT-L} returned by Algorithm FAST-L is also equal to S_{OPT-LU} if and only if it satisfies the extra constraint set given by (39). □

Claim 5. $\forall i \; \bar{\mu}_i > 0$ *implies* $\mu_i = 0$.

Proof. Assume that $\exists i$ such that $\bar{\mu}_i > 0$ and $\mu_i > 0$. In this case, (42) and (43) force us to choose $t_i = o_i$ and $t_i = 0$, which implies that $o_i = 0$. But, this is contrary to our assumption that $0 < o_i \; \forall i$ in the specification of the problem. □

Now, we are ready to justify line 5 of the algorithm.

Lemma 2. *If S_{OPT-L} violates upper bound constraints given by (39), then, in S_{OPT-LU}, $t_q = o_q \;\; \forall q \in \Gamma$.*

Proof. We will prove that the Lagrange multipliers $\bar{\mu}_i$, $i \in \Gamma$, are all nonzero, which will imply (by (42)) that $t_q = o_q$, $\forall q \in \Gamma$. Suppose $\exists m \in \Gamma$ such that $\bar{\mu}_m = 0$. We know that $\exists j$ such that $\bar{\mu}_j > 0$ (Proposition 4). Furthermore, $\mu_j = 0$ by Claim 5. Using (41), we can write $-\frac{1}{b_m} f'_m(t_m) - \frac{\mu_m}{b_m} = -\frac{1}{b_j} f'_j(t_j) + \frac{\bar{\mu}_j}{b_j}$, which gives (since $t_j = o_j$):

$$\frac{1}{b_m} f'_m(t_m) = \frac{1}{b_j} f'_j(o_j) - \frac{\bar{\mu}_j}{b_j} - \frac{\mu_m}{b_m} \quad (\bar{\mu}_j > 0,\ \mu_m \geq 0).$$

Since t_m is necessarily less than or equal to o_m, we can deduce $\frac{1}{b_m} f'_m(o_m) \leq \frac{1}{b_m} f'_m(t_m) < \frac{1}{b_j} f'_j(o_j)$, which contradicts our assumption that $m \in \Gamma$. □

Complexity: Notice that the worst case time complexity of each iteration is equal to that of Algorithm FAST-L, which is $O(n \cdot \log n)$. Observe that the cardinality of **F** decreases by at least 1 after each iteration. Hence, the number of iterations is bounded by n. It follows that the total time complexity $C_{OPT-LU}(n)$ is $O(n^2 \cdot \log n)$.

8 Conclusion

In this paper, we have addressed the periodic reward-based scheduling problem. We proved that, when the reward functions are convex, the problem is NP-Hard. Thus, our focus was on linear and concave reward functions, which adequately represent realistic applications such as image and speech processing, time-dependent planning, and multimedia presentations. We have shown that, for this class of reward functions, there always exists a schedule where the optional execution times of a given task do not change from instance to instance. This result, in turn, implied the optimality of any periodic real-time policy which can schedule a task set of utilization k on k processors. The existence of such policies is well-known in real-time systems community: RMS (with harmonic periods), EDF and LLF for uniprocessor systems, and, in general, any[4] scheduling policy that can fully utilize a multiprocessor system. We have also presented polynomial-time algorithms for computing the optimal service times. We believe that these efficient algorithms can also be used in other concave resource allocation/QoS problems such as the one addressed in [25].

We underline that, besides clear and observable reward improvement over previously proposed suboptimal policies, our approach has the advantage of not requiring any runtime overhead for maximizing the reward of the system and for constantly monitoring the timeliness of mandatory parts. Once optimal optional service times are determined statically by our algorithm, an existing (e.g., EDF) scheduler does not need to be modified or to be aware of mandatory/optional semantic distinction at all. In our opinion, this is another major benefit of having precomputed and optimal *equal* service times for a given task's invocations in reward-based scheduling.

In addition, Theorem 1 implies that, as long as we are concerned with linear and concave reward functions, the resource allocation can be also made in terms of *utilization* of tasks without sacrificing optimality. In our opinion, this fact points to an interesting convergence of *instance-based* [7], [21] and *utilization-based* [25] models for the most common reward functions.

About the tractability issues regarding the nature of reward functions, the case of step functions was already proven to be NP-Complete [21]. By efficiently solving the case of concave and linear reward functions and proving that the case of convex reward functions is NP-Hard, efficient solvability boundaries in (periodic or aperiodic) reward-based scheduling have been reached by our work (assuming P $\neq$ NP).

Finally, we have provided examples to show that the theorem about the optimality of identical service times per instance no longer holds if we relax some fundamental assumptions such as the deadline/period equality and the availability of the dynamic priority scheduling policies. Considering dynamic aperiodic task arrivals and investigating good approximation algorithms for intractable cases, such as step functions and error cumulative jobs, can be major avenues for reward-based scheduling.

4. Policies which model the time as slotted are an exception, as explained in Section 3.1.

Appendix

In Section 3, we mention that the total reward does not decrease by the transformation. Here, we will show that:

$$\sum_{j=1}^{q_i} f_i(t_{ij}) \leq q_i\ f_i(t'_i), \tag{46}$$

where $t'_i = \frac{t_{i1}+t_{i2}+\ldots+t_{iq_i}}{q_i}$ and the function f_i is concave. If f_i is a linear function of the form $f_i(t) = k_i \cdot t$, then $\sum_{j=1}^{q_i} f_i(t_{ij}) = k_i\,(t_{i1} + t_{i2} + \ldots + t_{iq_i}) = k_i\, q_i\, t'_i$ and (46) is immediately established.

If f_i is general concave, we recall that:

$$\alpha f_i(x) + (1-\alpha) f_i(y) \leq f_i(\alpha x + [1-\alpha] y) \tag{47}$$

$\forall x, y$ and for every α such that $0 \leq \alpha \leq 1$. In this case, we prove the validity of (46) by induction. If $q_i = 1$, (46) holds trivially. So, assume that it holds for $q_i = 1, 2, \ldots, m-1$. Induction assumption implies that:

$$\sum_{j=1}^{m-1} f_i(t_{ij})\ \leq\ (m-1)\, f_i\left(\frac{t_{i1} + \ldots + t_{i(m-1)}}{m-1}\right). \tag{48}$$

Choosing $\alpha = \frac{m-1}{m}, x = \frac{t_{i1}+t_{i2}+\ldots+t_{i(m-1)}}{m-1}, y = t_{im}$ in (47), we can write:

$$\begin{aligned} &\frac{m-1}{m}\, f_i\left(\frac{t_{i1} + \ldots + t_{i(m-1)}}{m-1}\right) + \frac{1}{m}\, f_i(t_{im}) \\ &\leq\ f_i\left(\frac{t_{i1} + \ldots + t_{im}}{m}\right). \end{aligned} \tag{49}$$

Combining (48) and (49), we get:

$$\frac{1}{m} \sum_{j=1}^{m} f_i(t_{ij})\ \leq\ f_i\left(\frac{t_{i1} + \ldots + t_{im}}{m}\right) \quad \forall m,$$

establishing the validity of (46).

Acknowledgments

Work done while Pedro Mejía-Alvarez was visiting the Department of Computer Science, University of Pittsburgh. A shorter version of this paper was presented at the 20th Real-Time Systems Symposium (RTSS '99). This work has been supported by the US Defense Advanced Research Projects Agency through the FORTS project (Contract DABT63-96-C-0044).

REFERENCES

[1] S. Baruah, "Fairness in Periodic Real-Time Scheduling," *Proc. 16th IEEE Real-Time Systems Symp.*, pp. 200-209, Dec. 1995.

[2] G. Bernat and A. Burns, "Combining (n,m) Hard Deadlines and Dual Priority Scheduling," *Proc. 18th IEEE Real-Time Systems Symp.*, pp. 46-57, Dec. 1997.

[3] A. Bertossi and L.V. Mancini, "Scheduling Algorithms for Fault-Tolerance in Hard-Real-Time Systems," *Real-Time Systems,* vol. 7, no. 3, pp. 229-245, 1994.

[4] M. Boddy and T. Dean, "Solving Time-Dependent Planning Problems," *Proc. 11th Int'l Joint Conf. Artificial Intelligence (IJCAI-89),* pp. 979-984, Aug. 1989.

[5] E. Chang and A. Zakhor, "Scalable Video Coding Using 3-D Subband Velocity Coding and Multi-Rate Quantization," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing,* pp. 574-577, July 1993.

[6] E. Chang and A. Zakhor, "Scalable Video Data Placement on Parallel Disk Data Arrays," *Proc. ISIT/SPIE Symp. Electronic Imaging Science and Technology,* pp. 208-223, Feb. 1994.

[7] J.-Y. Chung, J.W.-S. Liu, and K.-J. Lin, "Scheduling Periodic Jobs that Allow Imprecise Results," *IEEE Trans. Computers,* vol. 19, no. 9, pp. 1156-1173, Sept. 1990.

[8] W. Feng and J.W.-S. Liu, "Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines," *IEEE Trans. Software Eng.*, vol. 23, no. 2, pp. 93-106, Feb. 1997.

[9] W. Feng and J.W.-S. Liu, "An Extended Imprecise Computation Model for Time-Constrained Speech Processing and Generation," *Proc. IEEE Workshop Real-Time Applications,* pp. 76-80, May 1993.

[10] J. Grass and S. Zilberstein, "A Value-Driven System for Autonomous Information Gathering," *J. Intelligent Information Systems,* vol. 14, pp. 5-27, Mar. 2000.

[11] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines," *IEEE Trans. Computers,* vol. 44, no. 12, pp. 1443-1451, Dec. 1995.

[12] B. Hayes-Roth, "Architectural Foundations for Real-Time Performance in Intelligent Agents," *J. Real-Time Systems,* vol. 2, no. 1, pp. 99-125, 1990.

[13] E.J. Horvitz, "Reasoning under Varying and Uncertain Resource Constraints," *Proc. Seventh Nat'l Conf. Artificial Intelligence (AAAI-88),* pp. 111-116, Aug. 1988.

[14] J.K. Dey, J. Kurose, D. Towsley, C.M. Krishna, and M. Girkar, "Efficient On-Line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* pp. 217-228, May 1993.

[15] J.K. Dey, J. Kurose, and D. Towsley, "On-Line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks," *IEEE Trans. Computers,* vol. 45, no. 7, pp. 802-813, July 1996.

[16] G. Koren and D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips," *Proc. 16th IEEE Real-Time Systems Symp.*, pp. 110-117, Dec. 1995.

[17] R.E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence,* vol. 42, no. 2, pp. 189-212, 1990.

[18] C.M. Krishna and K.G. Shin, *Real-Time Systems.* New York: McGraw-Hill, 1997.

[19] K.-J. Lin, S. Natarajan, and J.W.-S. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems," *Proc. Eighth IEEE Real-Time Systems Symp.*, pp. 210-217, Dec. 1987.

[20] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[21] J.W.-S. Liu, K.-J. Lin, W.-K. Shih, A.C.-S. Yu, C. Chung, J. Yao, and W. Zhao, "Algorithms for Scheduling Imprecise Computations," *Computer,* vol. 24, no. 5, pp. 58-68, May 1991.

[22] D. Luenberger, *Linear and Nonlinear Programming.* Reading Mass.: Addison-Wesley, 1984.

[23] A.K. Mok, "Fundamental Design Problems of Distributed systems for the Hard Real-Time Environment," PhD dissertation, Massachusetts Inst. of Technology, 1983.

[24] M.J. Panik, *Classical Optimization: Foundations and Extensions.* Amsterdam: North-Holland, 1976.

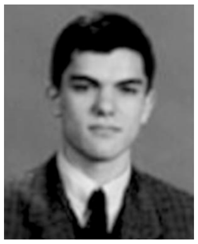
[25] R. Rajkumar, C. Lee, J.P. Lehozcky, and D.P. Siewiorek, "A Resource Allocation Model for QoS Management," *Proc. 18th IEEE Real-Time Systems Symp.*, pp. 298-307, Dec. 1997.

[26] W.-K. Shih, J.W.-S. Liu, and J.-Y. Chung, "Algorithms for Scheduling Imprecise Computations with Timing Constraints," *SIAM J. Computing,* vol. 20, no. 3, pp. 537-552, July 1991.

[27] C.J. Turner and L.L. Peterson, "Image Transfer: An End-to-End Design," *Proc. SIGCOMM Symp. Comm. Architectures and Protocols,* pp. 258-268, Aug. 1992.

[28] S.V. Vrbsky and J.W.S. Liu, "APPROXIMATE—A Query Processor that Produces Monotonically Improving Approximate Answers," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 6, pp. 1056-1068, Dec. 1993.

[29] S. Zilberstein and S.J. Russell, "Anytime Sensing, Planning and Action: A Practical Model for Robot Control," *Proc. 13th Int'l Joint Confs. Artificial Intelligence,* pp. 1402-1407, 1993.



Hakan Aydin is currently a PhD candidate at the University of Pittsburgh, Computer Science Department. He received BSc and MSc degrees in control and computer engineering from Istanbul Technical University in 1991 and 1994, respectively. He is a program committee member of the IEEE Real-Time Technology and Applications Symposium (RTAS 2001). His research interests include real-time systems, fault tolerance and low-power computing. He is a student member of the IEEE and a member of the ACM.

Rami Melhem received a BE degree in electrical engineering from Cairo University in 1976, an MA degree in mathematics and an MS degree in computer science from the University of Pittsburgh in 1981, and a PhD degree in computer science from the University of Pittsburgh in 1983. He was an assistant professor at Purdue University prior to joining the faculty of the University of Pittsburgh in 1986, where he is currently a professor of computer science and electrical engineering and the chair of the Computer Science Department. His research interests include real-time and fault-tolerant systems, optical interconnection networks, high performance computing, and parallel computer architectures. Dr. Melhem served on program committees of numerous conferences and workshops and was the general chair for the Third International Conference on Massively Parallel Processing Using Optical Interconnections. He was on the editorial board of the *IEEE Transactions on Computers* and served on the advisory boards of the IEEE Technical Committees on Parallel Processing and on Computer Architecture. He is the editor for the Plenum Book Series on Computer Science and is on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems*. Dr. Melhem is a fellow of the IEEE and a member of the ACM.

Daniel Mossé received a BS degree in mathematics from the University of Brasilia in 1986, and MS and PhD degrees in computer science from the University of Maryland in 1990 and 1993, respectively. He joined the faculty of the University of Pittsburgh in 1992, where he is currently an associate professor. His research interests include fault-tolerant and real-time systems, as well as networking. He has served on program committees for all major IEEE-sponsored real-time related conferences and as program chair for RTAS and the RT Education Workshop. He is currently writing a book on real-time systems. He is a member of the IEEE, the IEEE Computer Society, and of the ACM.

Pedro Mejía-Alvarez received his BS degree in computer systems engineering from ITESM, Queretaro, Mexico, in 1985, and a PhD degree in informatics from the Universidad Politecnica de Madrid, Spain, in 1995. He has been an assistant profesor in Seccion de Computacion CINVESTAV-IPN Mexico since 1997. In 1999, he held a research faculty position it the Department of Computer Science at the University of Pittsburgh and, in 2000, a visiting assistant professor position in the Department of Information Sciences and Telecommunications at the University of Pittsburgh. Previously, he held a researcher position at the Electrical Research Institute in Cuernavaca, Mexico, where he was involved in the design and implementation of a multiprocessor real-time operating system for a SCADA system for electrical substations. He is a fellow of the National System of Researchers of Mexico (SNI). His main research interests are real-time systems scheduling, adaptive fault tolerance, and software engineering.